Calhoun
Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1986

Microcomputer based linear system design tool

Wood, Roy L.

http://hdl.handle.net/10945/21975

Downloaded from NPS Archive: Calhoun

DUDLEY KNOX LIBRARY

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MICROCOMPUTER BASED LINEAR SYSTEM
DESIGN TOOL

by

Roy L. Wood, Jr.

September 1986

Thesis Advisor: George J. Thaler

Approved for Public Release; Distribution Unlimited

T233158

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for Public Release;
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	Distribution Unlimited
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION aval Postgraduate School	6b OFFICE SYMBOL (If applicable) Code 32
7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) onterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

MICROCOMPUTER BASED LINEAR SYSTEM DESIGN TOOL

12 PERSONAL AUTHOR(S)

Wood, Roy L. Jr.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986 September	182

16 SUPPLEMENTARY NOTATION

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Control Systems, Linear Systems, Computer-Aided Design, Computer-Aided Engineering, Modelling, Feedback Control

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The theory, algorithms, and operation of a continuous-time, linear control system design computer program is presented. The program, LCS-CAD, was developed to demonstrate automated transfer function block manipulation in conjunction with such classical control design techniques as Bode, Nyquist single and two-parameter root locus, and time domain response. Both numeric data and high-resolution graphs are available to the user. The software, which is completely interactive and menu driven, is written in structured Pascal to be run on the IBM-PC microcomputer.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
George J. Thaler	(408) 646-2056	Code 62TR

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for Public Release; Distribution is Unlimited

Microcomputer Based Linear System Design Tool

by

Roy L. Wood, Jr.
Lieutenant, United States Navy
B.S.C.S., Texas A & M University, 1980

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1986

ABSTRACT

The theory, algorithms, and operation of a continuous-time, linear control system design computer program is presented. The program, LCS-CAD, was developed to demonstrate automated transfer function block manipulation in conjunction with such classical control design techniques as Bode, Nyquist, single and two-parameter root locus, and time domain response. Both numeric data and high-resolution graphs are available to the user. The software, which is completely interactive and menu driven, is written in structured Pascal to be run on the IBM-PC microcomputer.

TABLE OF CONTENTS

I. LINEAR CONTROL SYSTEM COMPUTER AIDED DESIGN.....6

A. INTRODUCTION TO LCS-CAD.....6

B. LINEAR CONTROL SYSTEMS AND BLOCK DIAGRAMS.....6

C. PROGRAM FEATURES.....8

II. COMPUTER-AIDED SYSTEM DESIGN STEPS.....10

A. DEVELOPING A MODEL.....10

B. ANALYZING THE MODEL.....13

C. APPLYING THE SPECIFICATIONS.....25

D. SUMMARY.....31

III. DETAILED PROCEDURE MODULE DESCRIPTIONS.....32

A. PROGRAM OVERVIEW.....32

B. INPUT/CHANGE MENU ROUTINES.....34

1. Input Menu Hierarchy.....34

2. Input Utility Functions.....35

3. Blocks Record.....39

4. RootFinder Module.....41

5. Polynomial Expansion Routine.....44

6. Building an Equivalent Loop Block.....48

7. Input Procedure.....50

8. Change Procedure.....52

9. Add and Delete Blocks.....53

10. Save Blocks to Disk.....54

11. Retrieve Problem from Disk File.....55

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

12. Using Input Routines to Build Complex Functions.....56

C. LOCATION OF CHARACTERISTIC EQUATION ROOTS.....58

D. FREQUENCY ANALYSIS.....59

1. Nyquist Plot.....60

2. Bode Plot.....64

E. ROOT LOCUS.....65

F. TWO-PARAMETER ROOT LOCUS.....67

G. TIME RESPONSE.....75

H. UTILITIES.....80

I. HELP SCREENS.....81

IV. SUMMARY AND RECOMMENDATIONS.....83

APPENDIX A INPUT UTILITIES.....86

APPENDIX B PROGRAM LISTING.....94

LIST OF REFERENCES.....178

BIBLIOGRAPHY.....179

INITIAL DISTRIBUTION LIST.....180

I. LINEAR CONTROL SYSTEM COMPUTER AIDED DESIGN (LCS-CAD)

A. INTRODUCTION TO LCS-CAD

LCS-CAD is a software tool to aid in the analysis and design of continuous-time, linear control systems. It was designed to allow a user who is familiar with the "classical" design tools, such as Bode, Nyquist, and root locus, to apply these methods while eliminating much of the tedium.

The software system is completely menu-driven and attempts to be user-friendly in a number of ways. First, the hierarchical menu structure is only two levels deep at any point so the user will not become "lost" in the program. Second, user inputs are systematically verified and validated. Third, the user has access to a powerful "change" facility that allows erroneous system information to be changed easily. Finally, the program makes use of the concept of "transfer function blocks" for data entry and manipulation.

B. LINEAR CONTROL SYSTEMS AND BLOCK DIAGRAMS

The behavior of many physical systems can be described by linear differential equations, or at least approximated this way. Of these, many may be described as single-input, single-output systems. For example, simple electrical

circuits are often modelled as linear systems with single inputs, say V_{in}, and a single output, V_{out}. It is possible to think of these systems as "black boxes" being acted on by an input and producing an output. Inside these boxes, then, would be a function to translate the input "signal" to the output. This so called "transfer function" is very convenient when dealing with system models.

After the differential equations for a linear, time-invariant system with zero initial conditions are determined, the Laplace transform can be applied yielding algebraic equations in the complex variable "s". After some manipulation, the equations can be written as a ratio of output to input. This is the standard transfer function form.

Unfortunately, very large systems can have extremely complex transfer functions which are difficult to work with. Often, however, such problems can be divided into smaller sub-problems, and these sub-problems can be modelled as independent transfer function blocks. The blocks can then be analyzed separately, if necessary, and later recombined for overall system analysis. Recombination, or reduction, of transfer function blocks is done in accordance with the rules of "block diagram

algebra", or alternatively, by applying "Mason's rule".[1]

LCS-CAD is a loop-based system. That is, the program operates on a single path or closed loop at a time. This simplifies the computations required to automatically reduce the loop's blocks to single equivalent block transfer function. Since the reduction is done automatically, the user is free to concentrate on analysis and design and not the tedious process of manipulating block transfer functions.

C. PROGRAM FEATURES

LCS-CAD is a large program and requires at least 256K of memory to run. It will run on any IBM-PC or compatible "MS-DOS" computer and requires a standard IBM Color Graphics Adapter (CGA). It will run on either monochrome or color monitor, but the menus are color coded and are easier to work with if a color monitor is available. The graphics are in high-resolution (640x200) mode and only appear in white-on-black. The graphics can be dumped to an IBM-Graphics, Epson, or compatible printer.

The program is written in Turbo Pascal (tm) and, due to its limitation to 64K code and data segment sizes, LCS-CAD was compiled as a single main executable program and six "chain" files. The chain files are essentially programs compiled without a run-time support module. They,

---

[1] These block reduction techniques are described in detail in virtually all elementary control theory textbooks.

therefore, are not themselves executable. These program modules include:

(1)	CAD.COM	{the executable main module}
(2)	INPUT.CHN	{the input and change routines}
(3)	FREQ.CHN	{Bode and Nyquist modules}
(4)	TIMERESP.CHN	{the time response simulation module}
(5)	TWOPARAM.CHN	{the two parameter root locus module}
(6)	UTILMENU.CHN	{the utilities menu and routines}
(7)	HELPMENU.CHN	{the help menu and all help screens}

In addition to these files, there are two additional "system" files needed to run the program, 4x6.FON, and ERROR.MSG. The .FON file is a graphics lettering font used by the graphics routines to print letters and numbers on-screen. The ERROR.MSG file is a Turbo Pascal file containing various error messages.

The next chapter presents a detailed example using the LCS-CAD program to analyze and design a controller for a simple motor. From that discussion, many of the features of the program can be examined. The third chapter details each individual program module and outlines the theory and algorithms used to construct the LCS-CAD program. Chapter Four summarizes the work done already in LCS-CAD and proposes additions and further work which would make the program more useful and versatile.

II. COMPUTER-AIDED SYSTEM DESIGN STEPS

A. DEVELOPING A MODEL

The first step in the successful design of a linear system is to formulate a mathematical model which describes the system. Once modelled, the system can be analyzed and a suitable control system can be designed.

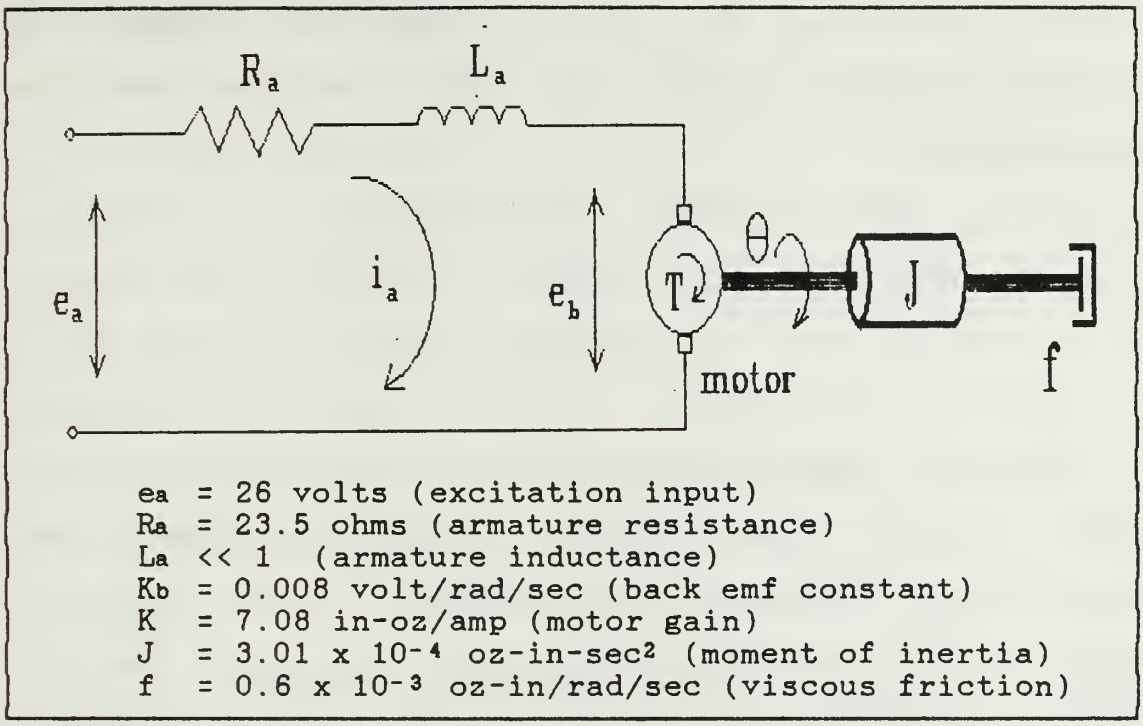
Consider, for example the simple armature controlled dc motor driving a load as shown in Figure 2-1. The left side of the figure shows the armature circuit including its resistive and inductive components. The mathematical model for this part of the motor can be derived from Kirchhoff's circuit law. The mechanical side of the arrangement, shown on the right, includes the motor generated torque, the load inertia, and the viscous damping effect. Newton's laws will provide the basis for modelling the mechanical portion of the motor/load combination.

The differential equation for the armature circuit is:

$$L_a \frac{di}{dt} + R_a\, i + e_b = e_a \qquad (2.1).$$

Similarly, the description of the mechanical system is a differential equation in shaft angle, but this time of second order.

$$J \frac{d^2\theta}{dt^2} + f \frac{d\theta}{dt} = T \qquad (2.2).$$



e_a = 26 volts (excitation input)
R_a = 23.5 ohms (armature resistance)
L_a << 1 (armature inductance)
K_b = 0.008 volt/rad/sec (back emf constant)
K = 7.08 in-oz/amp (motor gain)
J = 3.01×10^{-4} oz-in-sec^2 (moment of inertia)
f = 0.6×10^{-3} oz-in/rad/sec (viscous friction)

Figure 2-1. Schematic Diagram of the Armature Controlled DC Motor

Knowing that for an armature controlled dc motor the field current, i_f, is constant and, therefore, the magnetic flux is constant, the motor torque is directly proportional to the armature current

$$T = k i_a \qquad (2.3).$$

Also, under constant flux conditions, the back electromotive force, e_b, will be directly proportional to the motor shaft's angular velocity

$$e_b = k_b \frac{d\theta}{dt} \qquad (2.4).$$

These equations are sufficient to mathematically represent the dynamical behavior of the dc motor. If zero

initial conditions are assumed, then the differential equations can be written in their Laplace transform representation:

$$(Js^2 + fs)\,\theta(s) = T(s) = KIa(s) \qquad (2.5).$$

$$(Las^2 + Ra)\,Ia(s) + Eb(s) = Ea(s) \qquad (2.6).$$

$$Kb\,s\theta(s) = Eb(s) \qquad (2.7).$$

With the armature voltage as input and the motor shaft angle as output, the transfer function relation of the system can be formulated as shown below

$$\frac{\theta(s)}{Ea(s)} = \frac{K}{s\,[\,LaJs^2 + (Laf + RaJ)\,s + Raf + KKb\,]} \qquad (2.8).$$

Since the typical armature inductance of this type of dc motor is very small, it will be neglected. This greatly simplifies the transfer function without inducing much error due to the approximation as can be seen in Equation 2.9 below.

$$\frac{\theta(s)}{Ea(s)} = \frac{Km}{s\,(\,Tms + 1\,)} \qquad (2.9)$$

where $Km = K / (Raf + KKb)$ Motor gain constant

$Tm = RaJ / (Raf + KKb)$ Motor time constant

Using the numerical values for the motor given in Figure 2-1 the transfer function becomes

$$\frac{\theta(s)}{Ea(s)} = \frac{100}{s\,(\,0.1\,s + 1\,)} \qquad (2.10)$$

B. ANALYZING THE MODEL

Now that the system has been modelled, the next step is to examine the model's behavior before a controller is designed and installed. This step will tell the designer whether the system will require stabilization or simply "fine-tuning" to meet the design specifications. The computer can be used to assist in the analysis process.

The first step, of course, is to enter the model into the computer program. Working from the system's block diagram, shown in Figure 2-2, we begin the computer program. From the opening menu of LCS-CAD, shown here in Figure 2-3, we choose the "Input/Change Transfer Function(s)" option to bring us to the secondary menu shown in Figure 2-4.

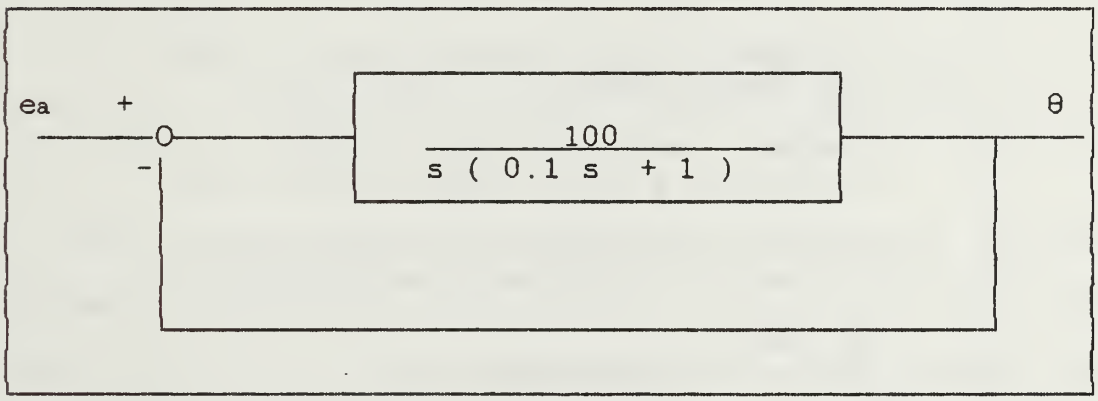


Figure 2-2. DC Motor Equivalent Block Diagram

```
          *** MAIN  MENU ***

<I> Input/Change Transfer Function(s)

<L> Location of Char. Eq. Roots
<F> Frequency Analysis
<R> Root Locus Analysis

<P> Two-Parameter Root Locus
<T> Time Response
<U> Utilities

<H> Help

<Q> Exit Program

        Press Your Selection
```

Figure 2-3. Main Menu

```
      *** INPUT/CHANGE MENU ***

<I> Input Block Transfer Function(s)
<C> Change Block in Current Loop

<A> Add a Block to Current Loop
<D> Delete a Block from Current Loop

<S> Save Current Loop to Disk File
<R> Retrieve Problem from Disk File

<H> Help
<Q> Exit to Main Menu

        Press Your Selection
```

Figure 2-4. Input/Change Menu

To initially enter the transfer function of the motor, select the <I> Input Block Transfer Function(s) option. The next screen will prompt for number of blocks in the current loop (only one in the case of the motor), and whether the block will be input from the keyboard or from a disk file. Since this is the initial input for this problem, it must be from the keyboard. Next, the block input page, as shown below in Figure 2-5, will appear.

```
                        Block Input

Block 1:
Is block in Forward (F) or Feedback (B) Path?


        What is the order of the Numerator?              0

        What is the order of the Denominator?            2

        What is the block gain constant?               100

        Will you enter the block in Factored (F)
                          or Coefficient (C) form?       C
```

Figure 2-5. Block Input Screen

The options here allow input of both forward path and feedback path blocks in any order. An input block can have numerator and/or denominator polynomial up to ninth order. There is provision for overall block gain input, and finally, the user is allowed to input the transfer function for the block in either coefficient or factored form.

The limitation of a block to order nine is artificial here, and systems up to thirtieth order can be input by breaking the block transfer functions into smaller order blocks. For example, if the transfer function were

$$\frac{(s^2 + 4s + 10)(s + 2)^2}{s^5 (s^4 + 2s^3 + 6s^2)(s^2 + 1)}$$

then the function could be divided into two or more blocks as shown here

$$\frac{(s^2 + 4s + 10)}{(s^4 + 2s^3 + 6s^2)} \quad x \quad \frac{(s + 2)^2}{s^5(s^2 + 1)}$$

with each block less than order nine and system order less than thirty.

The next screen shown will vary depending on whether the input is to be in factored or coefficient form. Figure 2-6 shows the factored input screen, and Figure 2-7 shows the coefficient form input screen.

```
                ***Block Transfer Function Input***
DENOMINATOR Transfer Function Input -- FACTORED Form
s = __________  +j __________
s = __________  +j __________


Press <F1> to change previous entry
```

Figure 2-6. Factored-Form Input Screen

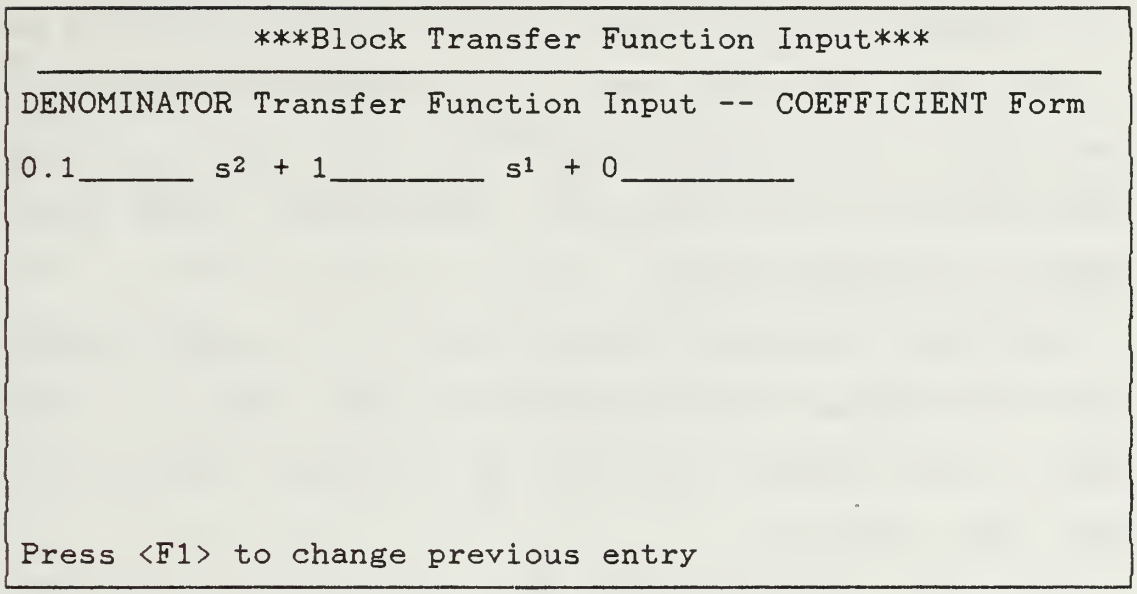

Block Transfer Function Input
DENOMINATOR Transfer Function Input -- COEFFICIENT Form
0.1 s² + 1 s¹ + 0
Press <F1> to change previous entry

Figure 2-7. Coefficient-Form Input Screen

For the dc motor example, the transfer function lends itself more easily to coefficient form input, and Figure 2-7 shows what the screen should look like after the entry is made. Notice that on both screens, the bottom line directs the user to press the F1 function key to move back to the previous entry. This facility can greatly enhance the capability to edit previous entries which may have been input incorrectly. Another error correction facility in the LCS-CAD program is the Change option executed from the Input/Change Menu (see Figure 2-4). This option will allow the user to select which block to change, then brings back the block input screens in the sequence discussed above except that the previously entered numbers are shown and a second message-prompt appears at the bottom of the screen.

This message directs the user to press the F10 function key to edit any item on the page. In addition to changing only the numbers previously entered, the user can also change the structure of the block by changing the order of the numerator and denominator.

Now that the system transfer function has been entered into the program, the analysis can begin. First, a quick check of the system's stability can be made. From the Main Menu, the selection

<L> Location of Char. Eq. Roots

will provide a listing of the unity-feedback closed loop characteristic equation roots. If all these roots are in the left-half of the s-plane, i.e., if all roots have negative real parts, then the system will be stable. As can be seen from Figure 2-8, the dc motor and load are stable.

```
    *** Block Transfer Function Closed-Loop Roots ***

ZEROS:

POLES:
s[1] =  - 5.000 +j  -31.225   s[2] =  - 5.000 +j   31.225


Press any key to continue or [Shift] [PrtSc] for hardcopy
```

Figure 2-8. Root Locations

Frequency domain analysis can be accomplished with LCS-CAD by selecting the

<F> Frequency Analysis

choice from the Main Menu. This selection will enable the user to prepare either a Nyquist (polar) diagram, or Bode (logarithmic) plot with user-selected radian frequencies. Figure 2-9 shows the on-screen selections for the Nyquist plot, and Figure 2-10 is the Bode selection screen. Both ask for the range of frequencies to be plotted. The first frequency should be an even power of ten. The range of frequencies is calculated based on the number of decades of frequency the user requests. That is, if the user selects 0.01 as the starting frequency with 4 decades, the end frequency will be 100 radians/sec.

```
          ***Bode/Nyquist Plotting Routine***
 ____________________________________________________

Bode (B) or Nyquist (N) Plot?              N
See the BIG (B) picture, or select your Window (W) ?  W
Open (O) or Closed (C) Loop Plot?          O

What is the first frequency to be plotted?
      (e.g. .01, .001, 1000, etc.)          .01


How many decades do you want plotted?   5
```

Figure 2-9. Nyquist Plot Parameter Selection

```
          ***Bode/Nyquist Plotting Routine***
          ___________________________________

Bode (B) or Nyquist (N) Plot?              B

Open (O) or Closed (C) Loop Plot?          O

What is the first frequency to be plotted?
     (e.g. .01, .001, 1000, etc.)          .1

How many decades do you want plotted?      3_
```

Figure 2-10. Bode Plot Parameter Selection

The "BIG picture" choice for the Nyquist simply causes the program to automatically select a broad range of frequencies and use a large scale plot. This can be a good initial selection if the user does not know how large a window will be needed to show the plot. The user may also choose an open or closed loop plot. If closed loop is chosen, a negative unity-feedback path is added around the overall block diagram, an equivalent transfer function is calculated, and the corresponding diagram is drawn.

For the dc motor example, both the Nyquist plot and the Bode diagram are shown in Figures 2-11 and 2-12 respectively.

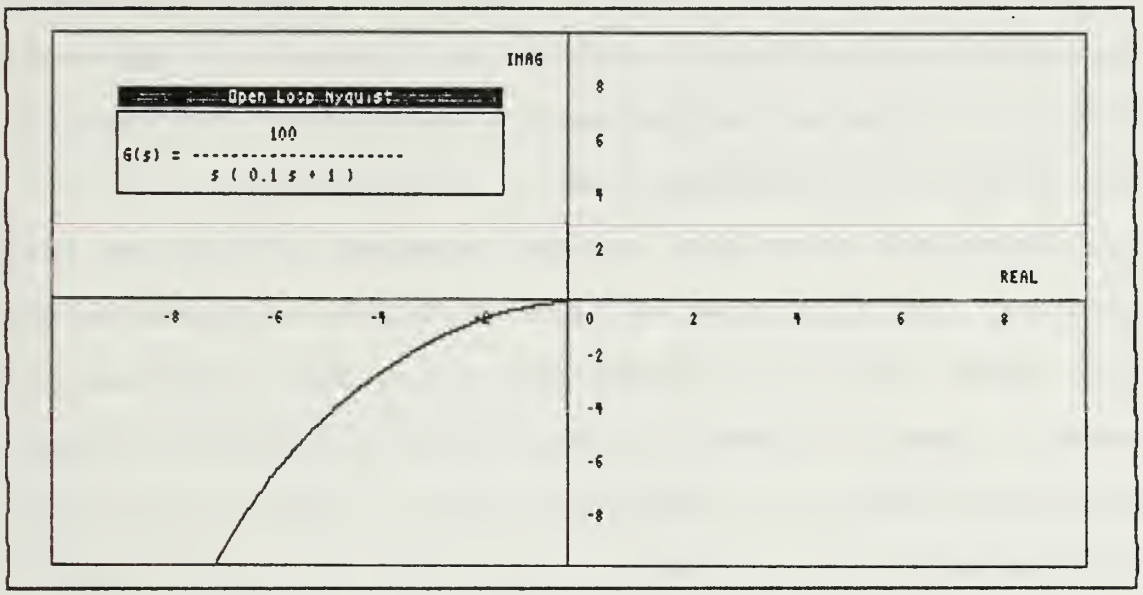


Figure 2-11. DC Motor Nyquist Plot

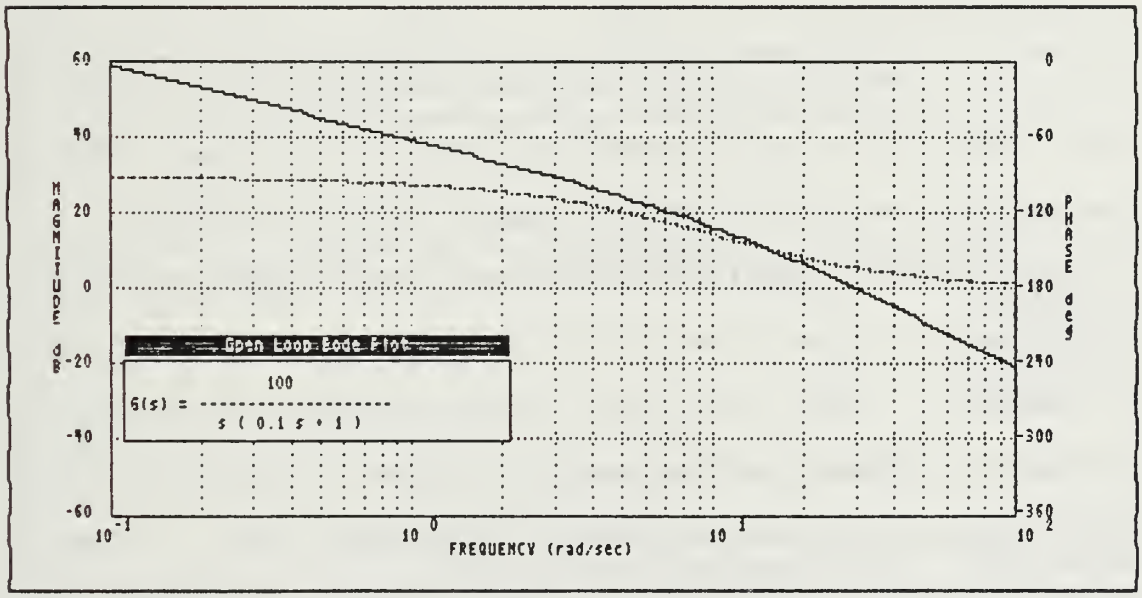


Figure 2-12. DC Motor Open-Loop Bode Diagram

From the Bode diagram it can be seen that the phase margin is low, only about ten degrees. The gain crossover frequency is near thirty radians per second. This means that the uncompensated system will be stable, but probably will have a long settling time.

Next, the root locus can be examined. With two real roots, it may be expected that with increasing gain, the roots will converge along the real axis, then separate and move in opposite directions parallel to the imaginary axis. To verify this, the program is used to generate the root locus graph.

From the Main Menu the selection now will be

<R> Root Locus

and will produce the screen shown in Figure 2-13.

```
          *** ROOT LOCUS PLOTTING ROUTINE ***

What STARTING value for variable gain do you wish?: .001

What ENDING value for variable gain do you wish?:      1

   *** VIEWING COORDINATES FOR ROOT LOCUS GRAPH ***
X-Minimum:     -10
X-Maximum:     1
Y-Minimum:     -20
Y-Maximum:     20

Positive or Negative Feedback? (P or N):     N

Any changes to these parameters? (Y or N):  N_
Press <F1> to change previous entry
```

Figure 2-13. Root Locus Parameter Input Screen

The program requests starting and ending values for the variable gain. This gain is multiplied by the block gain for the system, that is, for the dc motor if variable gain of 0.001 to 1 is chosen, then the actual gain the system will be varied through will be 100 x .001 to 100 x 1, or 0.1 to 100. The program computes the unity-feedback closed loop transfer function for an increment of gain then calculates and plots the location of all the roots. This process continues through the user-supplied range of gains.

The user is also asked to provide maximum and minimum values for X and Y plot axes. Knowing that the closed loop roots (with unity variable gain) are at -5.000 ± j 31.225, the X axis values of -10 to 1 can be expected to provide an adequate window. Y axis values of ± 20 may be adequate to show the root behavior. Once the required values are entered, the program will begin computation and plot a graph like the one shown in Figure 2-14.

Finally in the analysis, it is desirable to see the system's response in the time-domain to a typical input. For a dc motor, such an input may well be a step. This so-called step response can be calculated and plotted with the LCS-CAD program as can the system response to a sinusoid, ramp, or impulse input. All these inputs have user-selectable amplitudes and if the ramp input is selected, the user may select slope and dc-offset. In the case of the sinusoid, the radian frequency may be selected.

Figure 2-15 shows the time response parameter input screen and Figure 2-16 shows the unit step response curve for the dc motor. It can be seen from the time response that the motor's response overshoots excessively and has a relatively long settling time (as anticipated from the Bode diagram), although the system is stable and has no steady-state error.

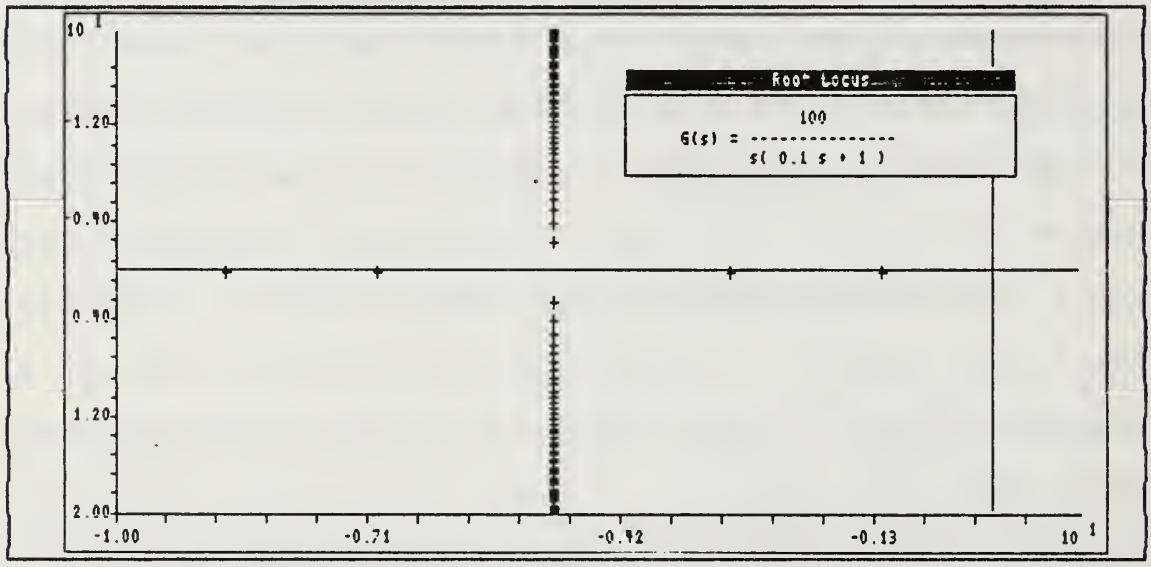


Figure 2-14. DC Motor Root Locus Diagram

```
          *** Time Response Plotting Routine ***

What is the input to your system?  STEP (S)
                                   RAMP (R)
                                   SIN WAVE (W)
                                   IMPULSE (I)      S

Input amplitude?   1

Open (O) or Closed (C) Loop simulation? C

How many seconds of simulation would you like to see?  2
```

Figure 2-15. Time Response Parameter Input Page

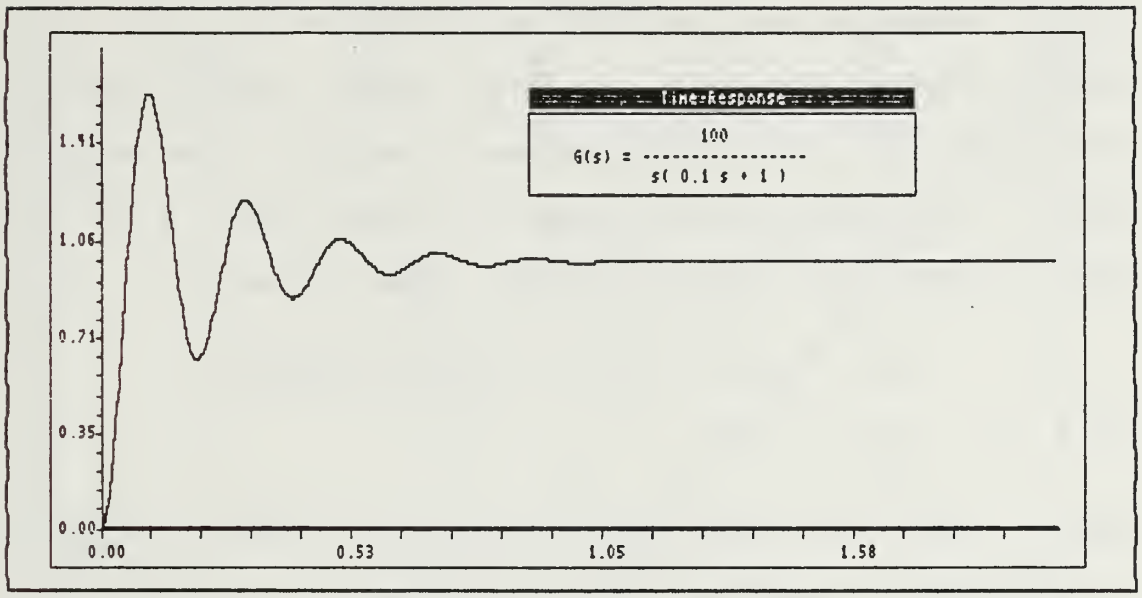


Figure 2-16. DC Motor Step Response

C. APPLYING THE SPECIFICATIONS

To use the motor in a larger system, the designer must either determine or be given the performance specifications for the motor. These specifications will usually outline the minimum acceptable performance the designer can tolerate from the system component.

For the example motor, suppose that the motor-controller combination must have a settling time of no more than 0.4 seconds and a first-overshoot of less than fifty-percent of the input signal when subjected to a step input. These constraints will allow demonstration of LCS-CAD in designing a simple cascade (series) compensator.

To assist the designer, curves for general second-order system response are available in virtually any basic control theory textbook. From these curves we can determine that for a fifty-percent overshoot, the required zeta(z), or damping factor, must be greater than 0.23. It then follows that a required phase margin Φ must be

$$\Phi_m = \tan^{-1} \left(2z / \sqrt{-2z^2 + \sqrt{4z^4 + 1}} \right)$$

or $\Phi_m \geq 25.9^\circ$.

Also, to meet the other specification of a settling time of 0.4 seconds, we need the relationship between settling time (t_s) and natural frequency (w_n). This connection is

$$t_s \leq 0.4 = 4 / z w_n \quad ,$$

or $w_n \geq 43.5$ rad/sec (since $z \geq 0.23$)

Since w_n is difficult to plot on the Bode graph, a more convenient frequency is needed. This is bandwidth frequency, or -3dB frequency, w_b.

$$w_b \geq w_n \sqrt{1 + 2z^2 + \sqrt{2 - 4z^2 + 4z^4}}$$

or $w_b \geq 65.1$ rad/sec.

Armed with the appropriate w_b, the compensator design can be attempted. On a large printout of the uncompensated Bode diagram as in Figure 2-17, which can be obtained using the "print screen" option {press the keys <Shift> and <PtrSc> (Print Screen) simultaneously}, locate the w_b point

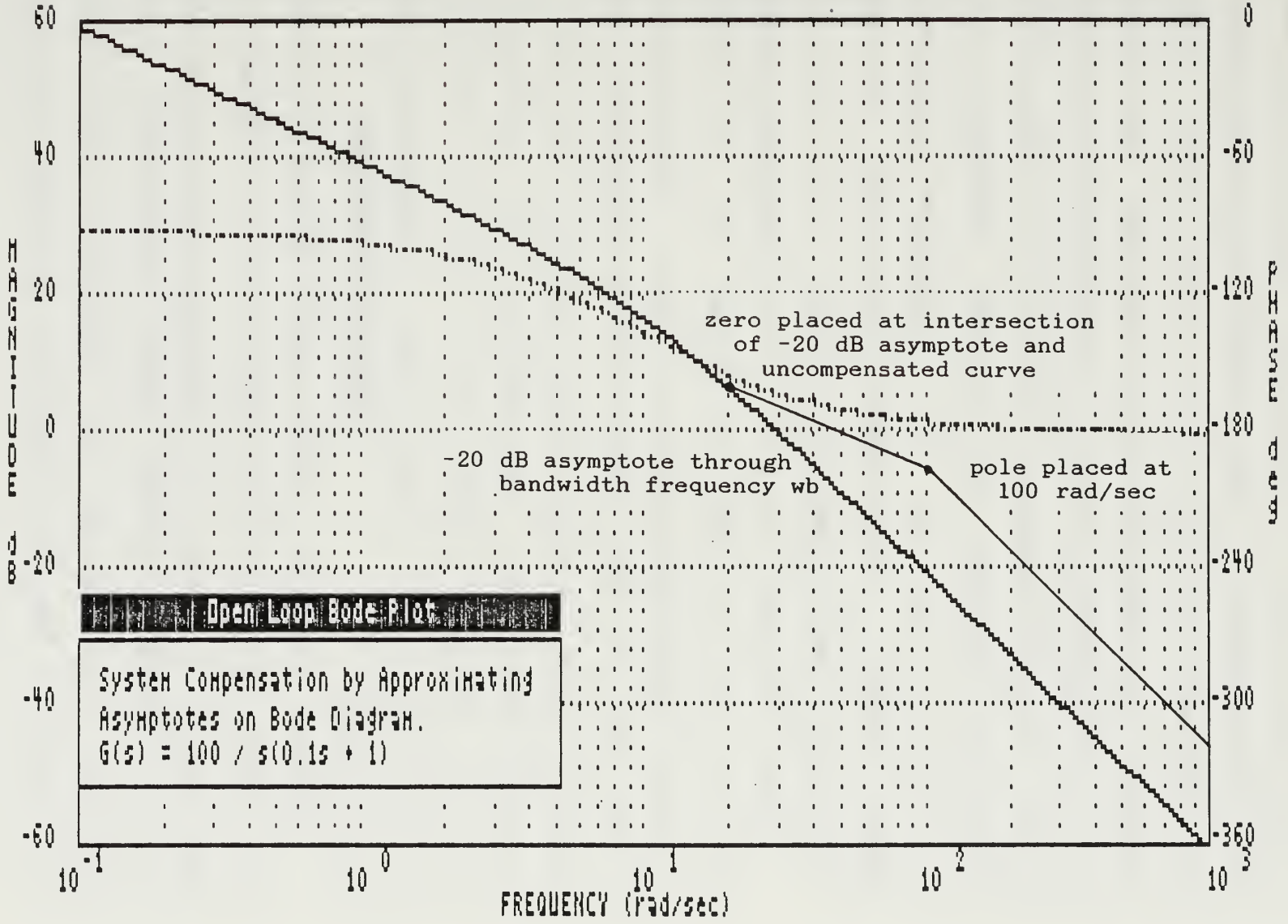


Figure 2-17. Working Bode Diagram with Asymptotes

and draw a -20 dB per decade asymptote through it. This will mark the crossover frequency and should provide good closed loop response. For simplicity, the compensator should only have one pole and one zero. This can be accomplished if the zero is located at the intersection point of the asymptote and the uncompensated system curve. As seen on the Bode diagram of Figure 2-17, the zero can be placed at w = 20 rad/sec. The pole can be placed at w = 100 rad/sec to give the -20 dB asymptote a reasonable length to ensure a large enough phase margin (Φm) and consequently, a good closed loop response.

Having determined pole and zero locations for the compensator, we can input the design into the LCS-CAD program and verify the solution quickly. From the Input Menu the option to "Add a Block" can be selected and the compensator input as block number two. In order not to change the motor gain, and therefore the steady state error, the compensator itself should have a an offsetting gain of 100/20 or 5.0. Once entered, system analysis can begin as before.

The open loop Bode diagram shown in Figure 2-18 shows that indeed the crossover frequency was increased to approximately 45 rad/sec and the phase margin to near 50 degrees. According to the earlier calculations, these parameters should ensure that the system is well within specifications. The root locus plot of Figure 2-19 also

helps to confirm this. As a final, conclusive check, however, the system response to a unit step, shown in Figure 2-20, can be seen to have a settling time of less than 0.2 seconds and a very small first overshoot. The design is, therefore, probably satisfactory. Since the compensated system performance is considerably better than the original specifications, it may be more costly to build than one that has slightly worse, but still satisfactory, performance. If this is the case, or had the first pole and zero placement not produced a satisfactory system, additional design trials could have been run quickly and easily with LCS-CAD.

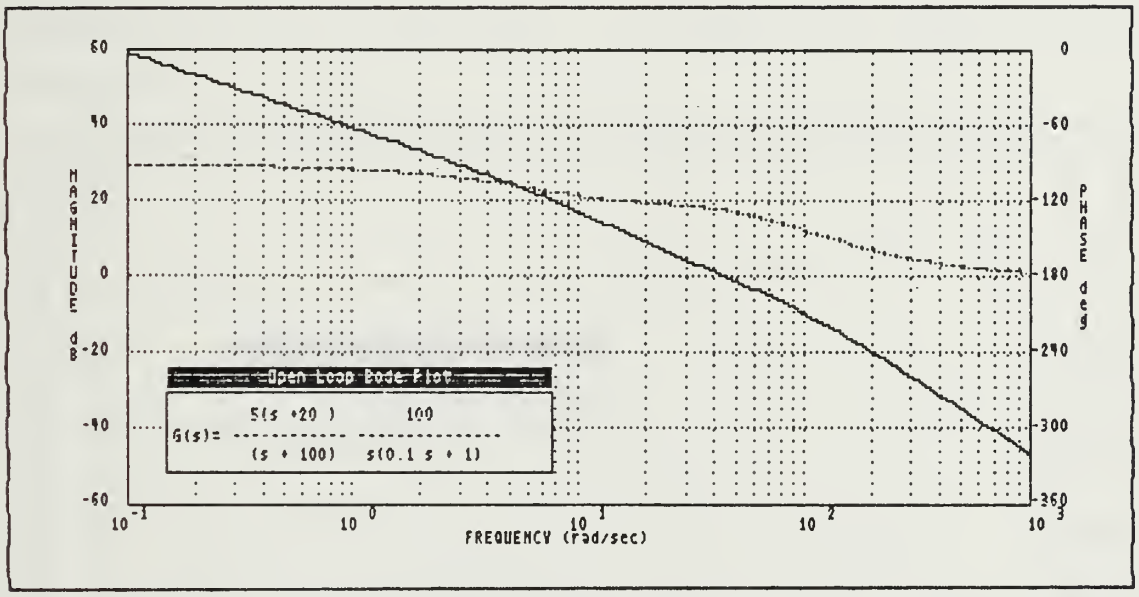


Figure 2-18. Compensated Open-Loop Bode Plot

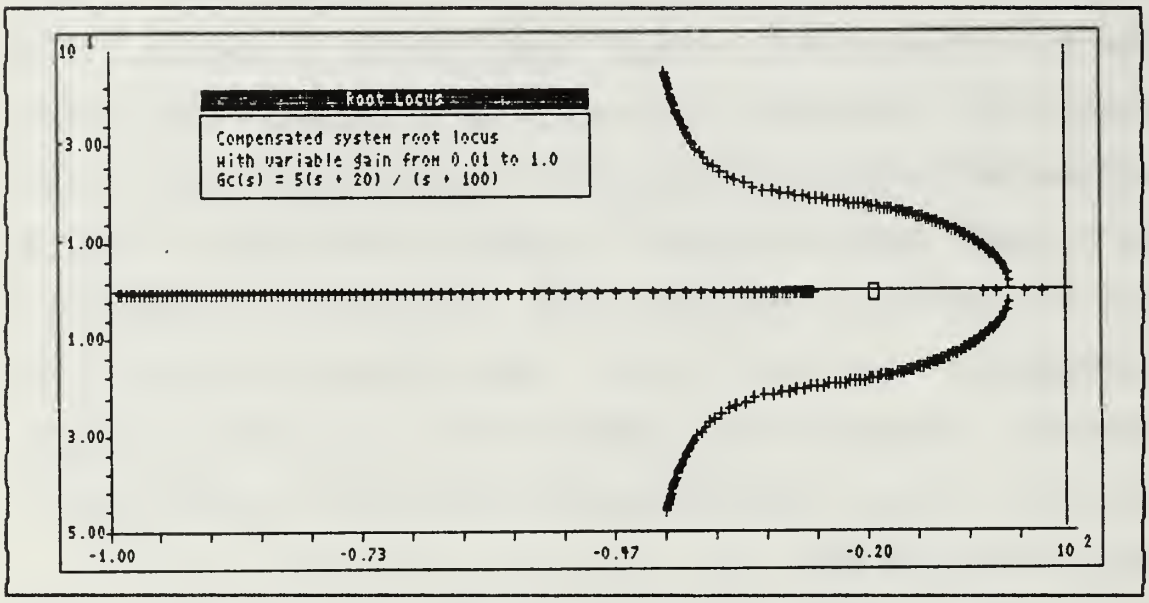


Figure 2-19. Compensated System Root Locus

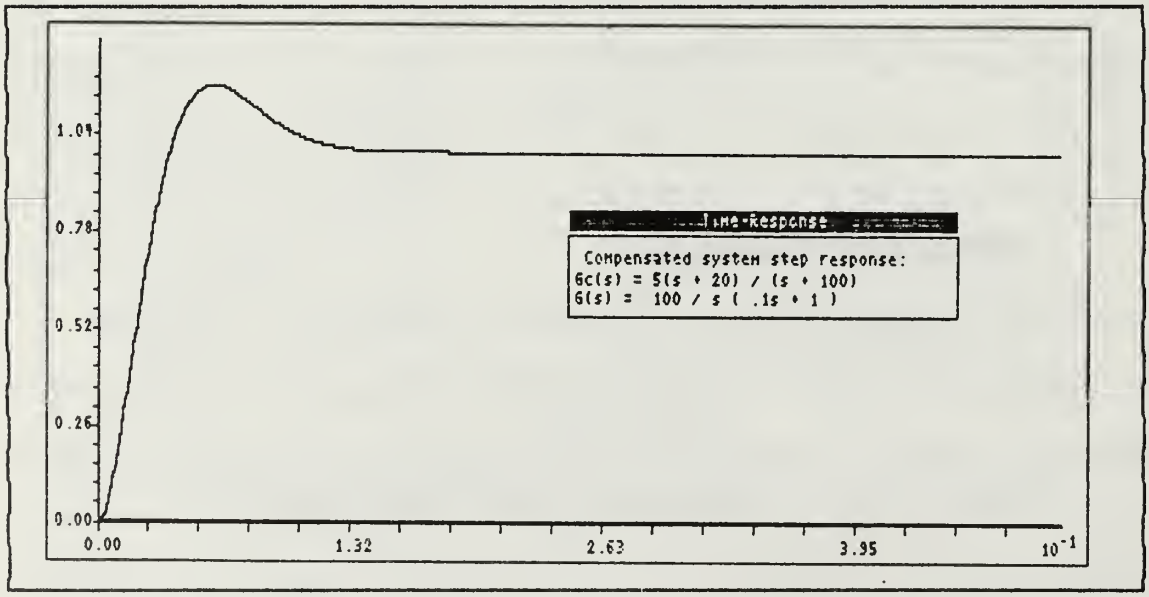


Figure 2-20. Compensated System Time Response

D. SUMMARY

In this chapter, a review of system design and an overview of LCS-CAD was provided. Several noteworthy program features are listed below.

* The Input/Change Routines enable the user to conveniently input and change transfer function block descriptions.
* The automatic block manipulator synthesizes the user's input blocks into an equivalent loop transfer function.
* The user has the ability to quickly and easily generate Bode, Nyquist, Root Locus, and Time Response plots.

Additional program features are available which were not discussed here, most importantly, the "two parameter root locus". This procedure as well as the other program features will be discussed in detail in the following chapter.

III. DETAILED PROCEDURE MODULE DESCRIPTIONS

A. PROGRAM OVERVIEW

LCS-CAD has a hierarchical, menu-based structure that allows the user general freedom to choose which design tool he wishes to use. The "Main Menu" is the starting point of the program and all other menus and utilities are available from here. In fact, the body of the main program, "CAD.PAS", simply calls the procedure "MainMenu" repeatedly until the user indicates that he is finished by typing <Q>.

The main menu procedure simply displays the menu shown in Figure 2-3, and provides for branching to other subroutines as requested by the user. A hierarchical representation of the Main Menu structure is shown in Figure 3-1 below.

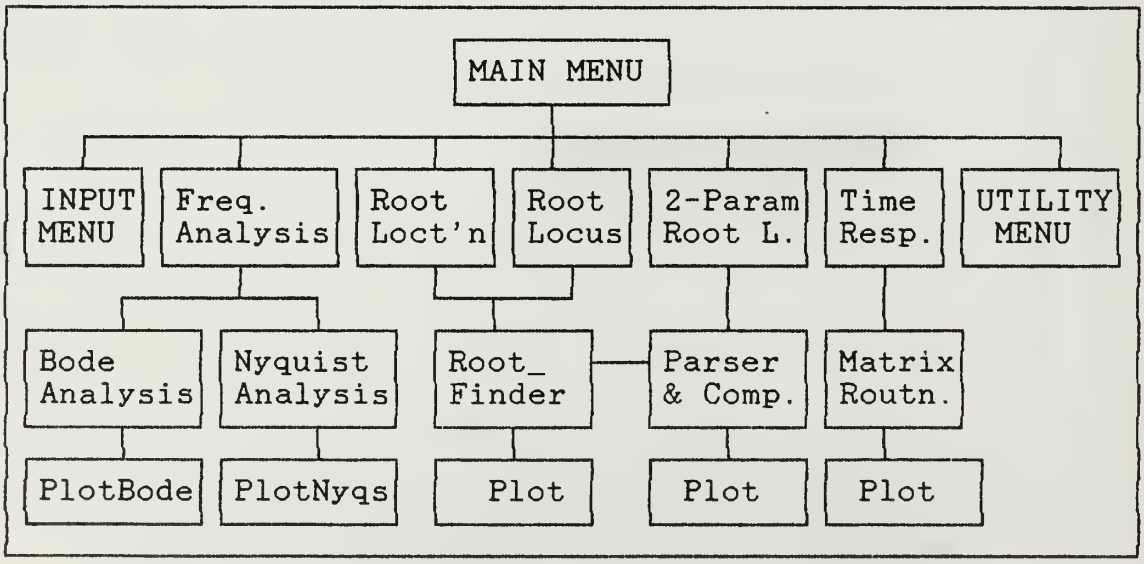


Figure 3-1. Main Menu - Functional Block Diagram.

As seen in Figure 3-1, the routine MainMenu can invoke two other menus, the Input Menu and the Utilities Menu. These will be discussed in detail later in this chapter. Also reachable directly from the Main Menu are the major analysis tools provided by the program, namely, frequency analysis, both single and two parameter root locus, and time response procedures. The frequency analysis portion of the program offers both Bode and Nyquist plots. The former represents system response as curves of phase (degrees) and magnitude (dB) versus radian frequency (logarithmic scale), and the latter displaying this information as a polar plot of imaginary and real parts of the magnitude.

The root locus programs include the classical "gain locus" where the system gain is varied over a user-selectable range. The two-parameter root locus allows the user to input the coefficients of a system's characteristic equation with two unknown parameters. The program then increments each of the parameters through ranges specified by the user and plots the resulting family of root locus curves.

The time response routine allows the user to select from a number of typical system inputs including step, ramp, impulse, and sinusoid as inputs to the current loop. The program then calculates the system's response to the input and plots it.

B. INPUT/CHANGE MENU ROUTINES

1. Input Menu Hierarchy

The input and change routines are a collection of procedures which allow the user to input and change the block transfer functions of his system. There are thirteen functional Pascal routines and numerous utility procedures which implement the input, change, and block manipulation functions. Figure 3-2 shows the hierarchy of these functional procedures within the larger input routine.

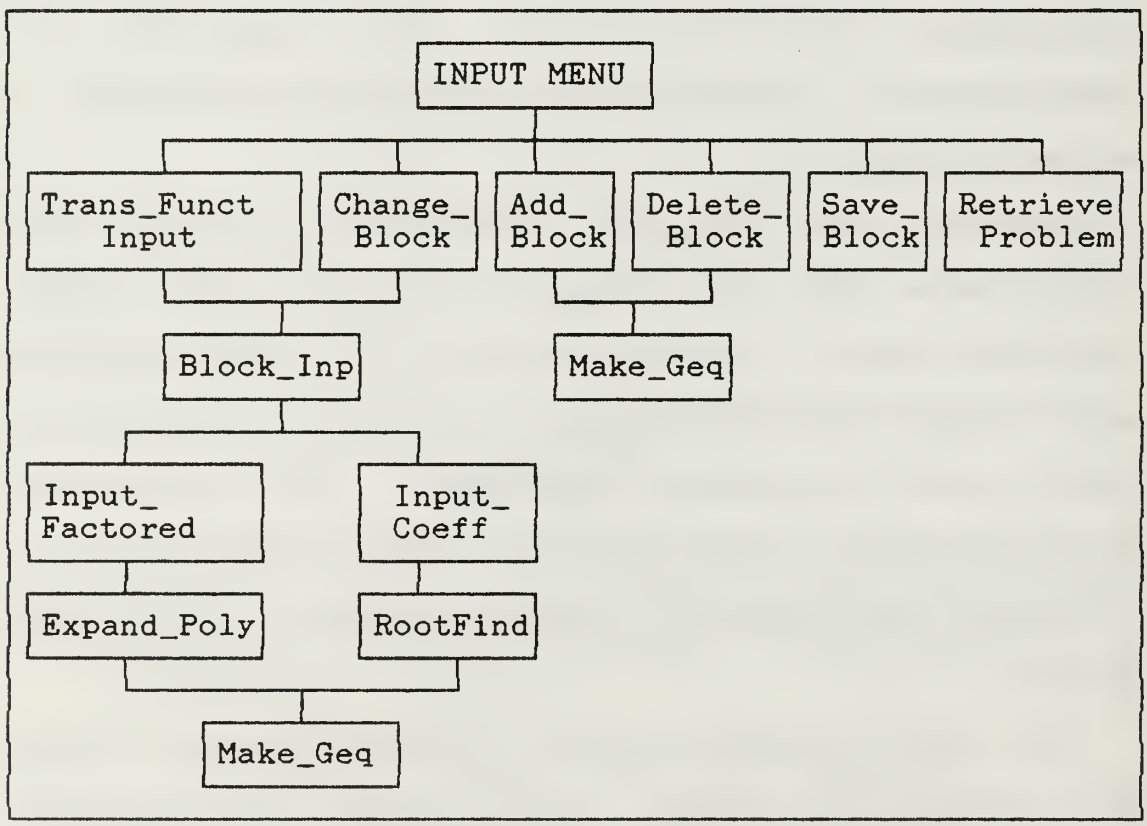


Figure 3-2. Input Routine Functional Block Diagram.

2. Input Utility Functions

One of the more difficult tasks in making a program "user-friendly" is validation of the user's input. This includes, but is not limited to, checking for the correct type of input (e.g., numbers, letters, or symbols), range checking numeric input, and checking to ensure that the answer makes "sense" (i.e., a "K" input does not satisfy a "Y" or "N" question). If the validation fails, an intuitive and graceful procedure must alert the user that a mistake has been made and re-prompt for a proper response. Some of these tasks have been accomplished in LCS-CAD through a group of very useful input routines available in the public domain. The routines used include a menu-generation procedure ("MainMenu"), procedures to write and center messages on the screen ("Msg" and "Center"), and two very powerful input routines called "Input" and "Input_Handler".

The procedure "Input" is a simple procedure which can be called by the programmer, but serves as the base routine for the more sophisticated routine "Input_Handler". Alone, Input will prompt for a single user-supplied entry, check it for type and length, and even provide a default answer if so programmed. Additionally, the routine provides the user with rudimentary text editing capabilities such as backspace, insert, and delete during

keyboard entry. The syntax for a call to Input is given below:

```
Input(type, default, col, row, length, uppercase, F1, F10);

  where  type = 'A' for alphanumeric,
               'N' for numeric,
               'F' for formatted (not used in LCS-CAD)
       default = text string to display default value
       col,row = column and row on screen for prompt
        length = number of character spaces in prompt field
     uppercase = boolean. True to convert input to all u.c.
        F1,F10 = boolean. True if function keys F1 or F10
                 pressed. Used only for Input_Handler
                 calls.
```

Type checking is automatically performed by the procedure and if the user attempts to input an alphabetic character in a numeric (N) field, for example, a "beep" will be generated to alert the user and the character will not be accepted. Once a valid input is supplied to the procedure and the <Return> key is pressed (signalling the end of user input), the routine returns the user's input in the global variable "Answer". Answer is always returned as a string-type variable, so if a number is expected, the programmer must provide for conversion to a numeric type via the built in Pascal function "val".

The procedure "Input_Handler" is a sophisticated, full-page, input editor. The programmer is required to predetermine each page lay-out and generate the necessary textual prompts. A set of array elements describing the desired input format is then generated in the form shown below:

```
P[1] := '2505A02501T010102'
  .
  .
P[n]   where n is limited to 40 elements.
```

The contents of each P[] element is a coded, column-dependent, descriptor field explained below:

Column Numbers	Description
1-2	column for input field to start
3-4	row for input field
5	input type - A : alphanumerics, N : numeric, F : formatted (not used), $: dollar (not used)
6-8	length of the input field
9-10.	element of the output global array Filvar[] to store the user's input
11	set to T if Caps Lock is to be set
12-13	default item number
14-15	prompt item number
16-17	validation number

Default, prompt, and validation numbers call the procedures "Get_Default", "Say_Prompt", and "Do_Validation" respectively. The procedures are primarily comprised of Pascal "case" statements that use the associated item numbers to perform some programmer defined default, prompt, or validation written into the case-statement. These procedures will be described more fully when discussing the LCS-CAD routines which use them.

Input_Handler can now be called with the syntax:

```
Input_Handler( '5-character string' );
```

where String Column Description

```
1 . . . . N : new entries
            C : changes to old entries
            D : re-display of entry (not used)
2-3. . . . first element of P[] array to use
4-5. . . . last element of P[] array to use
```

If calling the procedure with "N" for new entries, the contents of the corresponding output array element, "Filvar[]", is cleared and readied for new input. If "C" is used, the old value of the associated Filvar variable is displayed as the default input for the field. This is essential for full-screen editing of inputs. Input_Handler calls the Input routine as described above, but uses the boolean variables F1 and F10 for access to all fields on a screen. In the "N"- new entries mode, a user prompt is displayed in the lower left corner of the screen instructing the user to press F1 to edit the previous entry on the screen. If in the "C" - change mode, an additional prompt tells the user to press F10 to discontinue editing on that screen. By using the F1, <Return>, and F10 keys, the user can quickly and effectively edit previously entered screens of data to either change a problem or correct erroneous entries.

Input_Handler returns the user's input to the calling program in the global variable array "Filvar". As with the Input routine's Answer variable, all the Filvar elements are string-type and must be converted if numeric input is needed.

These utility routines were helpful in developing LCS-CAD and are used extensively throughout the program for user input. Several other subroutines are available in the package either for programmer use or to be called by the routines described here. Appendix A is a descriptive excerpt from the user's guide supplied with these subroutines.

3. Blocks Record

LCS-CAD uses the transfer function block as the basic foundation element of the program. Not only does this have the advantage of being intuitive to the designer accustomed to classical block manipulation design, but facilitates algorithm design which is also simple for the engineer to follow. Pascal handles this block-by-block design particularly well with its "record" structure. Each transfer function block in the user's current loop is described by one corresponding Pascal record. Each record is a logical grouping of all the parameters necessary to describe the block. This record structure is shown below in Figure 3-3.

```
BLOCKS = record
           NZeros, NPoles : integer;
           K              : real;
           RealPartZero,
           ImagPartZero,
           RealPartPole,
           ImagPartPole   : PolyArray;
           NumCoeff,
           DenCoeff       : PolyArray;
           LeadNumCoeff,
           LeadDenCoeff   : Real;
           FeedBack,
           Factored       : Boolean;
```

Figure 3-3. Blocks Record Structure

The record contains the order of the numerator and denominator polynomials in the variables NZeros and NPoles respectively. The coefficients of the polynomials are contained in the arrays (of type PolyArray) NumCoeff and DenCoeff. The factors of the numerator polynomial are held in the arrays RealPartZero and ImagPartZero for the real and imaginary portions of the complex conjugate factors respectively. Likewise, the denominator factors are stored in RealPartPole and ImagPartPole.

"LeadNumCoeff" and "LeadDenCoeff" are used to "normalize" and "un-normalize"the leading coefficients of the polynomials. That is, for calculations, the routines require polynomials whose leading coefficient is one, but for display, the polynomials must be converted back to the original input form. These two real variables facilitate those conversions.

"FeedBack" is a boolean variable that is true when the block is in the feedback path and false when it is in the forward path of the loop. "Factored" is true if the transfer function was initially input in factored form and false if the input was in coefficient form.

If the user inputs a block transfer function into the program in coefficient form, the routine "RootFinder" is called to generate the factors of the polynomials and store them in the appropriate record variables. In the case of factored input, the routine "Expand_Poly" provides the coefficients of the expanded polynomials and stores them in the record variables "NumCoeff" and "DenCoeff". RootFinder and Expand_Poly will be discussed later.

Once all the blocks in a loop have been input, the procedure "Make_Geq" is invoked which reduces the transfer functions of all the blocks in the loop into a single "equivalent" block with an identical record structure as all the other blocks. Most plotting functions and analysis tools use this equivalent block record for all their computations.

4. RootFinder Module

One of the most important routines in LCS-CAD is the procedure called "RootFinder". This sub-program's function is to find the factors, or roots, of any given polynomial. The procedure is used by the Input routine to find the poles and zeros of a transfer function input as a

quotient of two polynomials. It is also called by the root-locus and two-parameter root locus programs to find successive roots of the transfer function while changing the system gain or other parameters.

Procedure RootFinder uses Bairstow's method to find the roots of a polynomial with real coefficients numerically [Ref. 1]. This algorithm iteratively searches for a quadratic factor of a given polynomial and, when it finds the factor, deflates the original polynomial and repeats the process. The algorithm is outlined in Figure 3-4. Procedure RootFinder is called and passed the order of the polynomial (N), the polynomial coefficients (Coeff) in array form, the initial guesses for P and Q (P1 and Q1), and returns two arrays of real and imaginary roots (RealPartRoot and ImagPartRoot). RootFinder first normalizes the input polynomial to have a leading coefficient of one and loads the polynomial into an array A. The B and C arrays are initialized to contain all zeros.

The procedure checks the input polynomial order and handles the simple cases of a zero, first, or second order polynomial. If higher order, the procedure invokes the Bairstow algorithm and computes a quadratic factor. When a factor is found RootFinder calls a supplementary procedure named Solve_Quadratic to determine the complex conjugate roots using the classical quadratic equation. With the "B"

Given an n^{th} degree polynomial:

$$a_3 x^n + a_4 x^{n-1} + \ldots + a_{n+2} x + a_{n+3} ,$$

and the initial coefficients P and Q of the quadratic factor:

$$x^2 - Px - Q ,$$

then

```
Set B(1), B(2), C(1), and C(2) = 0.
DO WHILE DeltaP > epsilon (tolerance value), or
         DeltaQ > epsilon,
    DO FOR J = 3 to n + 3 step 1,
           Set B(J) = A(J) + P * B(J-1) + Q * B(J-2).
           Set C(J) = B(J) + P * C(J-1) + Q * C(J-2).
           Set DENOM = C(N+1)² - C(N+2) * C(N).
           IF DENOM = 0 THEN
              Set P = P + 1.
              Set Q = Q + 1.
              Repeat from beginning.
           ENDIF.
           Set DeltaP = [ -B(N+2) * C(N+1) + B(N+3)
                                   * C(N) ] / DENOM.
           Set DeltaQ = [ -C(N+1) * B(N+3) + C(N+2)
                                   * B(N+2) ] / DENOM.
           Set P = P + DeltaP.
           Set Q = Q + DeltaQ.
    ENDDO.
ENDDO.
```

Figure 3-4. Bairstow's Algorithm

array now holding the deflated polynomial of order n-2, theprocedure checks again for the simple low-order cases and solves the remainder of the problem or continues searching for another quadratic factor. To look for another quadratic factor, the B matrix coefficients are loaded into the A matrix and the entire procedure is repeated.

RootFinder operation is affected by two parameters which are somewhat arbitrarily selected. These are

Epsilon, the acceptable error, and IterationCount, the number of times the procedure will repeat the search for a factor. If Epsilon is chosen to be very small, and IterationCount very large, the accuracy of the solution should improve, but the execution time of the procedure will be degraded. Currently, Epsilon is 0.00001 and IterationCount is 40. In program tests against known results these two constants have provided satisfactory results without noticeable execution time degradation.

5. Polynomial Expansion Routine

The procedure Expand_Poly takes as input a set of real and complex conjugate factors and expands them into a polynomial with real coefficients. There are numerous algorithms to accomplish this, but the most intuitive one was selected for use in LCS-CAD. The process here follows closely the steps one would take if performing the operation long-hand. The steps are outlined in Figure 3-5.

The process is best explained by example. Assume that the following set of factors is to be operated on by the subroutine Expand_Poly algorithm.

$$(s + 1 + j1.414)(s + 1 - j1.414)(s + 4 + j0)$$

The first two factors constitute a complex conjugate pair since their real and imaginary parts have the same magnitude with the imaginary parts differing in sign. When these two factors are multiplied together, the

result will be a real-coefficient, quadratic polynomial. For this particular example the product of the conjugate pair will be

$$(s^2 + 2s + 3).$$

The program will recognize that, since there is a non-zero imaginary part of the first factor, a complex conjugate pair is present in the problem and resolve this first by forming the quadratic as above. To accomplish this, TEMP[1] will be set to $1^2 + 1.414^2$ or 3.0, and TEMP[2] will be set to (2 * 1) or 2.0, in accordance with the algorithm.

Assignments to POLY will be identical and the order of the system, originally three, will be reduced to one. This will leave only the (s + 4) factor to deal with in the subsequent step. So the program will examine this next factor and determine it to be real. HOLD[1] will be set to three and HOLD[2] will be set to zero. This initial setup is shown in Figure 3-6.

Now the polynomial in POLY is shifted left one place to simulate multiplying it by the unity coefficient of the s-term in the (s + 4) factor. Finally, the contents of TEMP are multiplied by HOLD[1] and added to the shifted POLY contents as seen in Figure 3-7.

The addition indicated in Figure 3-7 yields the polynomial with coefficients

$$\underline{1}s^3 + \underline{6}s^2 + \underline{11}s + \underline{12} .$$

These coefficients are stored back into the array TEMP and the process continues if there are more factors.

```
Given n real or complex factors of a polynomial:

      (s + a1 ± jb1)(s + a2 ± jb2)...(s + an ± jbn)

 where a's are real and jb's are imaginary or zero,
 then

 Set Temp[i] and Poly[i] = 0.  {for all i to n}
 order = n.

 IF b = 0 THEN     {real value}
    Set TEMP[1] = POLY[1] = a1.
    Set TEMP[2] = POLY[2] =  1.
   Decrement order by 1.
 ELSE
    Set TEMP[1] = POLY[1] = a1² + b1². {conjugate mult}
    Set TEMP[2] = POLY[2] = 2 * a1.
    Set TEMP[3] = POLY[3] = 1.
    Decrement order by 2.
 ENDIF.

 WHILE order > 0 DO
       IF b = 0 THEN
          Set HOLD[1] = ai.
          Set HOLD[2] =  0.
          Set POLY = ShiftLeft(POLY).
         Set POLY[1] = 0.
          Set POLY = POLY + (TEMP * HOLD[1]).
          Set TEMP = POLY.
          Decrement order by 1.
       ELSE
          Set HOLD[1] = ai² + bi². {conjugate multiply}
          Set HOLD[2] = 2 * ai.
          Set HOLD[3] = 1.
         Set POLY = ShiftLeft(POLY).
         Set POLY[1] = 0.
          Set POLY = POLY + (TEMP * HOLD[2]).
          Set POLY = ShiftLeft(POLY).
          Set POLY[1] = 0.
          Set POLY = POLY + (TEMP * HOLD[1]).
         Set TEMP = POLY.
          Decrement order by 2.
       ENDIF.
 ENDDO.
```

Figure 3-5. Expand_Poly Algorithm

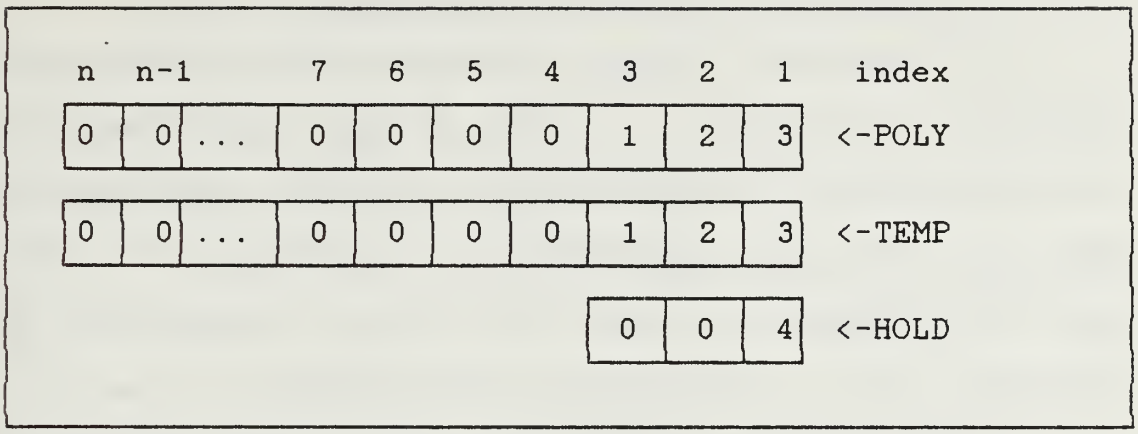


Figure 3-6. Initial Array Contents

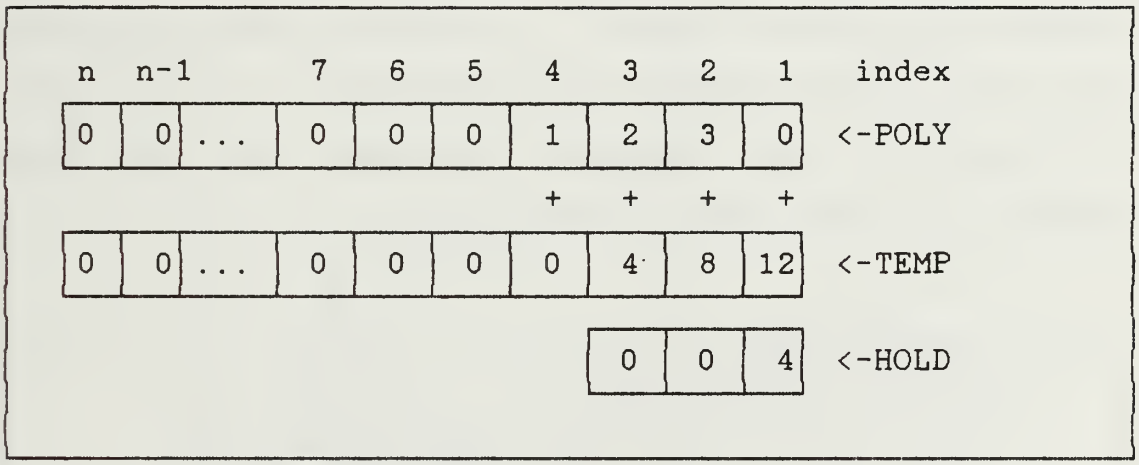


Figure 3-7. Array Contents After Third-Factor Operation

In the actual program implementation, the conjugate multiplication is developed as a separate procedure called "Conjug_Mult" for program readability. Also, the coefficients in the final result stored in the POLY array are reversed to conform to the established storage order protocol.

6. Building the Equivalent Loop Block

An essential element of the LCS-CAD program is the "Make_Geq" procedure. This routine calculates the equivalent block transfer function from all the blocks in the user input loop. Conceptually, the process is a simple one, and because of the block record structure, the algorithm, too, is relatively straightforward.

For a given loop each of the transfer function blocks will be in either the forward or feedback path. The first step will be to condense these blocks into single forward and feedback blocks. To combine blocks in the same path the transfer functions are multiplied, or more simply, the factors of the transfer functions are just lumped together as shown below in Figure 3-8.

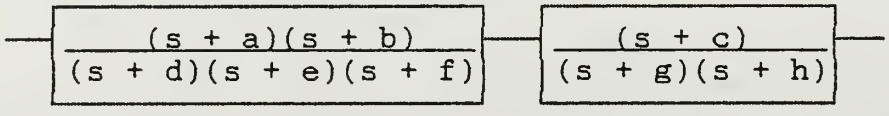


Figure 3-8a. Two Path T.F. Blocks

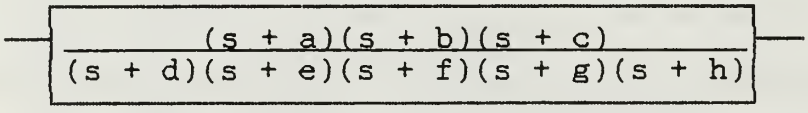
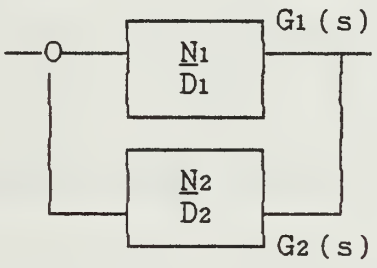


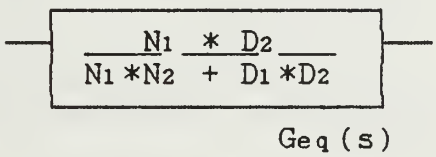
Figure 3-8b. Equivalent Block Diagram

Figure 3-8. Transformation of Two Path Transfer Functions to a Single Equivalent Transfer Function Block

Once the forward and feedback path blocks have been reduced to two equivalent blocks, they can easily be reduced to one single block by application of Mason's rule. This is demonstrated in Figure 3-9.

The Make_Geq procedure closely follows these steps to reduce the loop block to a single "G-equivalent" block. The simplified algorithm is shown below in Figure 3-10. In the actual implementation, additional arrays and temporary variables were required to hold the real and imaginary parts of each pole and zero, and the forward and feedback block gains.





If N's are numerator and D's are denominator polynomials, then by Mason's Rule the equivalent transfer function is:

$$Geq(s) = \frac{G1(s)}{1 + G1(s)G2(s)}$$

. {Multiplying through by D1 *D2 gives the right-hand block expression.}

Figure 3-9. Reduction of a Forward and Feedback Block to a Single Equivalent Block

```
Given NBlocks transfer functions

then

Initialize Fwd_Zero_Count, Fwd_Pole_Count,
           Feedback_Zero_Count, Feedback_Pole_Count = 0.

FOR i = 1 TO NBlocks STEP 1 do
  IF FeedbackBlock THEN
    COLLECT(Feedback_Zeros and _Poles) in Feedbck_Array.
    INCREMENT(Feedback_Zero_Count & _Pole_Count).
  ELSE
    COLLECT(Forward_Zeros and _Poles) in Forward_Array.
    INCREMENT(Forward_Zero_Count & _Pole_Count).
ENDFOR.

WITH G_eq DO
 {construct Geq Zeros}
 COLLECT(Forward_Array_Zeros & Feedbck_Array_Zeros) in
                                                Geq_Zeros.
 {construct Geq Poles}
 COLLECT(Forward_Array_Zeros & Feedbck_Array_Zeros) in
                                                    Temp1.
 EXPAND(Temp1) {to Polynomial and store} in TempPoly1.

 COLLECT(Forward_Array_Poles & Feedbck_Array_Poles) in
                                                    Temp2.
 EXPAND(Temp2) {to Polynomial and store} in TempPoly2.

 Set Geq_Denominator_Coefficients to TempPoly1+TempPoly2
 FACTOR(Geq_Denominator_Coefficients) in Geq_Poles.
ENDWITH.
```

Figure 3-10. Make_Geq Procedure Algorithm

7. Input Procedure.

The basic input program relies on four separate procedures, namely, Trans_Function_Input, Block_Input, Input_Factored, and Input_Coeff. Trans_Function_Input is the procedure invoked from the Input Menu. It serves two functions:

(1) Query the user for number of blocks in the loop, then execute repeat a call to Block_Input until all the blocks are entered,

(2) Call the Make_Geq procedure after all blocks are input to form the equivalent transfer function.

The Block_Input procedure is the heart of the input routine. It queries the user for block attributes like number of poles and zeros, block gain, and whether the user will input the block transfer functions as factors or polynomial coefficients. Block_Input relies heavily on the services provided by the procedures Input and Input_Handler described earlier. These utility procedures give the user a great deal of flexibility with their built-in error checking and validation and the full-screen editing capabilities.

Based on the user's choice of factored or coefficient form input, Block_Input calls Input_Factored or Input_Coeff respectively. These routines display user prompts for their particular kind of input and also use the Input_Handler routine extensively.

An unusual segment of the program implementation worthy of note involves the coefficient form of input. If the leading coefficient input by the user is not unity, then the program will normalize the polynomial. This involves dividing the polynomial through by the leading coefficient, then correcting the block gain to compensate for the change. Several algorithms in LCS-CAD, in particular the Root_Finder procedure, expect a unity

leading coefficient on a block polynomial. The normalization process does produce unwanted side-effects, however. If the user decides to view or change a transfer function that was entered in coefficient form, then he could be confused by the changes to block gain and the normalized polynomial. To alleviate these problems, two variables for each block, LeadNumCoeff and LeadDenCoeff, hold the original values of the user input leading coefficients of the numerator and denominator polynomials respectively. If the user chooses to change the block, the normalization process can be reversed and the coefficients and gain restored to their input values.

8. Change Procedures.

One of the most important features of the LCS-CAD user interface is its ability to allow changes to the block descriptions quickly and easily. The major tool which supports this facility is the Input_Handler procedure discussed earlier. The LCS-CAD input and change procedures expand Input_Handler's full screen editing support to include the capability to change the actual structure of a previously entered block. That is, the user can change the order of the block transfer function as well as the coefficients or factors that were previously entered. This very powerful combination facilitates both input error correction and changes made during the design process.

To make these changes possible, the procedure Change_Block sets a boolean variable called "Change" to true when the procedure is entered. This alerts the Block_Input procedure, which is also used for initial user input, that changes are to be made to the block whose index is passed as a parameter. The Block_Input routine then reinitializes the appropriate Filvar[] array elements to the previously entered values and uses them as default values to each query (see Section 2. Input Utility Functions for Filvar explanation). The process then continues exactly as for first-time input.

9. Add and Delete Blocks

The procedure Add_Block allows the user to add a block to the current loop. The routine simply increments the block counter variable, NBlocks, and calls the block input procedure, Block_Input. On return from Block_Input, a new G-equivalent is calculated.

Delete_Block is a slightly more complicated procedure. It must first ask the user to identify the block number of the block to delete, check that the block exists, and then remove it from the current loop. The block is effectively removed from the loop by deleting its block index number and adjusting the remaining block indices. A new G-equivalent block is computed after a block is deleted.

10. Save Blocks to Disk

The procedure used to store a problem to a floppy or hard disk file is Save_Block. It queries the user for drive designator (A through D), and filename for the loop blocks to be saved. Only the first eight characters of an MS-DOS filename are allowed; the program appends a filename extension of ".BLX" to each loop. This extension is used to later to limit the disk search for legitimate block files when retrieving the data. Once a drive and filename are supplied the procedure opens the file and stores the loop data. The data file is a sequential file of records stored as illustrated in Figure 3-11.

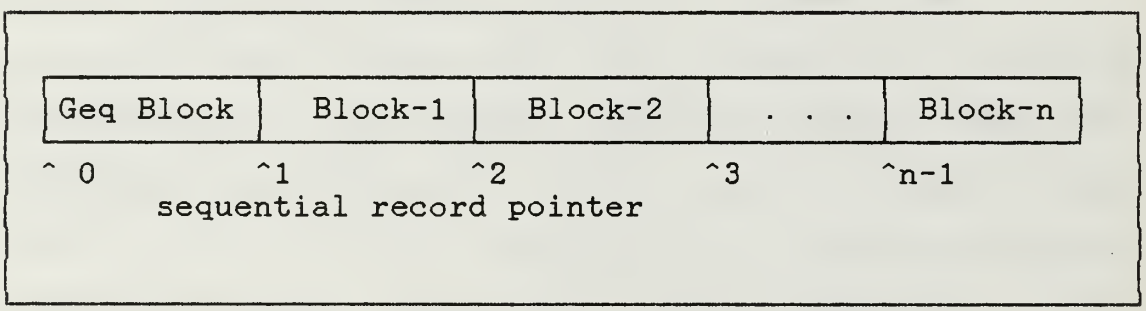


Figure 3-11. Block Storage Scheme

Note from Figure 3-11 that the equivalent block, Geq, is stored first at the zero file-pointer position. This is convenient since, regardless of the number of blocks in the loop, the equivalent block description is always at position zero and easy to access. There are two opportunities to read this file during the program, the

first available from the single block input routine when the program asks whether the block will be read from a file or from the keyboard. If the file option is taken, then only the Geq block is accessed. The other time the file can be read is from the "Retrieve Problem from Disk File" option on the Input Menu. This option will read all the blocks into memory, including the Geq block. This option is discussed in detail in the next section.

11. Retrieve_Problem_from_Disk_File

Like the Save_Block procedure, the Retrieve_Problem procedure first queries the user for the data drive where the block descriptions are located. Once this is done, the program calls another public domain procedure called "Directory". This routine uses MS-DOS function calls to query the disk drive and read the directory listing. It then displays the directory in a window on screen with a moveable cursor activated with the arrow keys on the numeric keypad. The cursor can be moved to point to the desired file and, when the <Return> key is pressed, the filename is returned to the Retrieve_Problem procedure. Retrieve_Problem then opens the file and reads the contents.

The Directory procedure will only request the disk filenames with the MS-DOS extension ".BLX". This eliminates the possibility that the user can attempt to

read in a block that is not a data block (recall that the Save_Block procedure automatically appended this extension onto the user-supplied filename).

12. Using Input Routines to Build Complex Systems

The tools described thus far should be sufficient to build arbitrarily complex systems if the proper approach is used. The powerful block manipulator is capable of reducing a loop with up to nine blocks. Block size is limited to a ninth order numerator or denominator, due to screen display limits, and overall system size is limited to a polynomial of thirtieth order, based on the compiler-imposed memory size limit. The greatest limitation of the LCS-CAD program for the designer is the ability to only handle one loop in memory at one time. This can be overcome by working a complex problem "from the inside-out".

Given a complex problem like the one shown in Figure 3-11, the problem can best be solved by the following steps:

(1) Enter block B and C as a system and save to disk;

(2) Enter block D from the keyboard and the BC equivalent from the diskfile. Save result to disk;

(3) Enter blocks A, E, and F from keyboard and the BCD equivalent from diskfile. Analyze the system.

If this were a typical feedback design problem, the engineer would need to analyze the interior loops for stability before proceeding to the outer loops as a

standard practice, so the inside-to-out limitation would be minimized. At any rate, this limitation is at worst an annoyance and should have no serious effect on overall design performance.

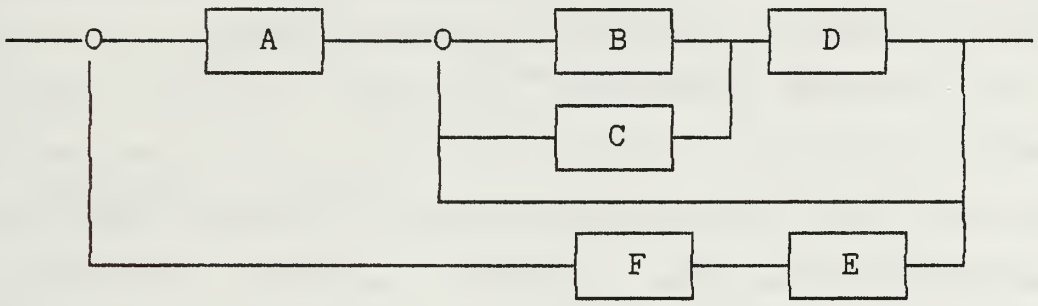


Figure 3-11a. Complex Block Diagram

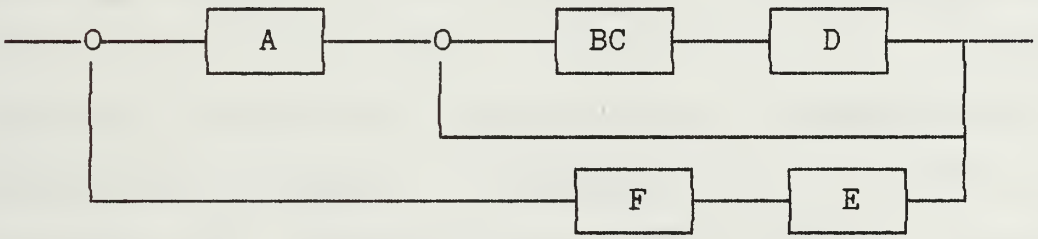


Figure 3-11b. Reduced BC Loop

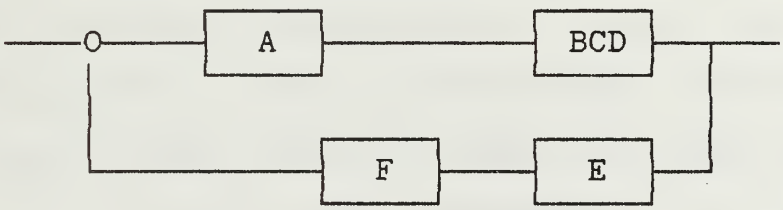


Figure 3-11b. Reduced BCD Loop

Figure 3-11c. Equivalent Block.

Figure 3-11. Reduction of a Complex Block Structure with LCS-CAD

C. LOCATION OF CHARACTERISTIC EQUATION ROOTS

As discussed in Chapter 2, one way to quickly check the stability of a system is to examine the closed-loop roots of the system's characteristic equation. If all the roots are in the left-half of the S-plane, i.e. real parts of the roots are negative, then the system is stable.[2]

The extreme usefulness of such a utility was the impetus for placing it in the Main Menu. Once the user has input his block diagram and returned to the main menu, only a keystroke is required to check the loop for stability. If the loop is not stable, then the user knows immediately that compensation is required and can begin the design process.

The procedure which computes and displays the closed-loop roots is called "ShowRoots". Negative unity feedback is used to close the loop on the system. The algorithm is shown in Figure 3-12. Recall that in a unity feedback system, the closed-loop denominator polynomial is equal to the sum of the open-loop denominator coefficients and the block gain times the numerator coefficients, since

$$G_{c1}(s) = \frac{G_{o1}(s)}{1 + G_{o1}(s)} \quad .$$

The closed-loop zeros are identical to the open-loop zeros.

---

[2] It can be established for a stable, causal system that the roots will all be in the left-half of the S-plane. Since, from an engineering standpoint, all physically realizable systems will be causal, therefore this generalization can be made.

```
Given the G-equivalent block for the system:

CALCULATE the on-screen display position.
DISPLAY Geq.NumCoeff ROOTS.{C.L. zeros = O.L. Zeros}

SET ClosedLoopPoly = (Geq.NumCoeff * K) + Geq.DenCoeff.
CALL RootFinder.
DISPLAY ROOTS of ClosedLoopPoly.
```

Figure 3-12. ShowRoots Algorithm.

D. FREQUENCY ANALYSIS

Frequency-domain analysis is essentially the examination of a system's response to input sinusoids of varying frequency. Not only can this so called "frequency response" can be computed analytically from the system transfer function, but the model of a complex physical system can be obtained experimentally by injecting input sinusoids and observing the output resonse. For these reasons, frequency domain design techniques have been developed and used for many years.

Among the most popular design tools in the frequency-domain are graphical techniques known as Bode and Nyquist plots. Historically, the graphical methods allowed the engineer to visualize the system's behavior. He could then apply various heuristic methods and approximations to obtain a qualitative "best guess" at how the system would respond to a given input. The other alternative of deriving the quantitative response was often mathematically complex and time-consuming if it was possible at all.

Now with the advent of digital computers, and especially the desktop "personal computer", the engineer has a device that can provide the quantitative information while still providing the familiar graphical tools that he is accustomed to designing with. The end result should be faster, easier, and more accurate designs.

1. Nyquist Plot

The Nyquist plot is a graph of the magnitude of G(jw) versus the phase angle of G(jw) as w is varied through a range of frequencies. The graph is plotted in polar coordinates. For each frequency, the phase and magnitude is calculated and plotted on the Nyquist plot. Figure 3-13 shows a graphical interpretation of magnitude and phase.

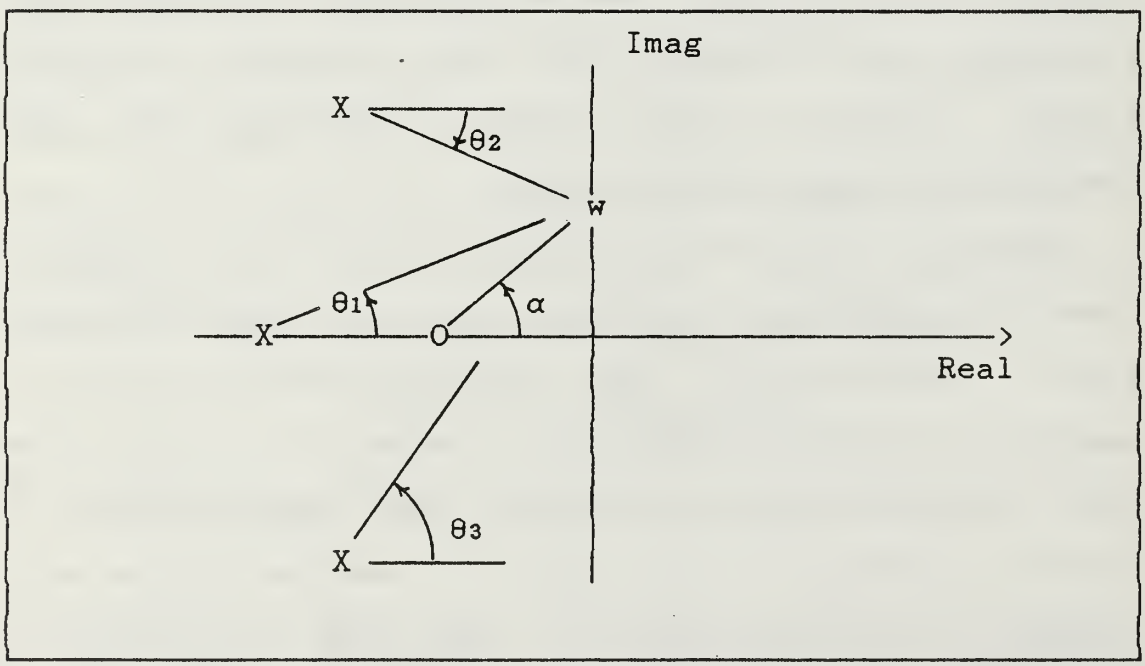


Figure 3-13. System Magnitude and Phase

For computation purposes the magnitude of a single pole or zero can be written as

$$Magn = \sqrt{(RealPart)^2 + (w - ImagPart)^2}$$

and its phase is

$$Phase = \tan^{-1}[\ (w - ImagPart) / RealPart\ ].$$

For multiple poles or zeros, the magnitude and phase are simply computed for each individual pole or zero, then the magnitudes multiplied and the phases and added together.

The the overall system magnitude can be computed as

$$Magn_{system} = Magn_{zeros} / Magn_{poles}$$

and the overall system phase is

$$Phase_{system} = Phase_{zeros} - Phase_{poles}.$$

The LCS-CAD procedure Bode performs precisely these calculations to compute both the Bode and Nyquist system response. The algorithm is outlined in Figure 3-14. Once the system phase and magnitude are computed, the simple transformation into cartesian coordinates is required to plot the information.

The user is given several options to select from regarding the plotting and calculation parameters for the Nyquist plot. The beginning frequency can be selected as can the number of decades to be plotted. The user can choose to manually select a window size (maximum and minimum values on the plotting scale) or can alternately choose to see the "big picture". This option will plot a

Given the poles and zeros of a transfer function:

$$G(s) = \frac{(s + a_1 \pm j\, b_1)(s + a_2 \pm j\, b_2)\ldots(s + a_m \pm j\, b_m)}{(s + c_1 \pm j\, d_1)(s + c_2 \pm j\, d_2)\ldots(s + c_m \pm j\, d_m)}$$

and a range of frequencies w_1 .. w_k , then

```
WHILE w < wk DO
 SET ZMagn = PMagn = 1.0; SET ZPhase = PPhase = 0.0;
 FOR i = 1 TO m STEP 1 DO   {compute for zeros}
     SET ZMagn = ZMagn * SQRT( ai2 + (w - bi)2 ).
     SET ZPhase = ZPhase + ArcTan((w - bi)/ -ai ).
 ENDFOR.
 FOR i = 1 TO n STEP 1 DO   {compute for poles}
     SET PMagn = PMagn * SQRT( ci2 + (w - di)2 ).
     SET PPhase = PPhase + ArcTan((w - di)/ -ci ).
 ENDFOR.
 SET Magnsys = ZMagn/PMagn. SET Phasesys = ZPhase-PPhase.
 SET PlotX = Magnsys * cosine(Phasesys ).
 SET PlotY = Magnsys * sine(Phasesys ).
 INCREMENT w.
ENDWHILE.
```

Figure 3-14. Nyquist Algorithm

range of frequencies from 10^{-3} radians/second to 10^{+5} radians/second and automatically plot the resultant Nyquist graph on a 50 x 50 scale. The user may also choose to plot open-loop or closed-loop response. If the open-loop response is desired, the G-equivalent block is used. If the closed-loop response is selected, then the procedure described in Section C. is used to compute the equivalent unity-feedback, closed-loop system transfer function prior to calculating the Nyquist response.

The program implementation consists of two procedures, "Bode" and "Plot_Nyquist". As described above,

the procedure Bode doubles as both the Bode and Nyquist calculation procedure. The routine Plot_Nyquist actually implements the graphics which plots the Nyquist numbers generated in the Bode subprogram.

Within the Plot_Nyquist routine is a call to the procedure "Graph_Menu". Graph_Menu is called by all procedures which produce a graph. It provides a pop-up menu offering the user the opportunity to add a title to the graph just plotted, print the graph on a printer, print the numbers used to generate the graph, or quit and return to the Main Menu.

If the user elects to title the graph, the plot is swapped onto a "virtual screen" in memory and a text screen appears offering the user three blank lines to type in his title. When the title is completed the plot is returned tothe physical screen and a window is drawn with the title text inside. Since there is no way to tell a priori where on the screen the important part of the plot will be located, the title block can be relocated anywhere on the screen by using the cursor arrow keys. When the title box is where the user wants it, the <Return> key is pressed thus freezing its position and recalling the graph options menu to the screen. The option which allows the user to print the graph will first remove the menu options block from the graph and dump the remaining screen information to

the printer. If the user decides to instead print the numbers, the procedure is slightly more complicated.

The "dump numbers" mode saves the current graphics screen and offers the user a choice to print the data to the printer or a pre-selected data filename on disk. For most of the graphics routines, the number of points is too large to store in a matrix when they are computed to plot the graph. Therefore, when the option to print the numbers is selected, the same computations are repeated again, and the numbers are printed or stored to a disk file rather than used to plot the graph. The user should be cautioned that because of the large number of points calculated and plotted, the length of the printer listing can be excessive. If only a few data points are of interest, the print to a file option is the better choice. Then the user can scan that file with a word processor or the MS/DOS "type" command and examine the points of interest.

2. Bode Plot

Most of the features of the Bode algorithm has been discussed in the above explanation of the Nyquist plot. The major difference between the Bode and Nyquist routines is the manner in which the information is displayed. The Bode graph displays two separate plots of magnitude and phase versus radian frequency. The magnitude plot is converted to decibels using the relation

$$Magn_{dB} = 20\log_{10}(Magn)$$

and the phase is converted to degrees by computing

$$Phase_{deg} = (180/\pi)(Phase).$$

The frequency is plotted on a logarithmic scale along the abscissa. The plots of magnitude and phase are superimposed on the same graph with the ordinate values of -180° phase and 0dB magnitude aligned. This is convenient for the designer when reading gain and phase margin from the graph. Since phase margin is read at the zero crossover of the magnitude curve and gain margin is read at the -180° crossover of the phase curve, these values can be read directly without shifting either curve up or down.

The program implementation is much like that used for the Nyquist procedure. The Bode routine calculates the numbers required for the plots and the procedure "Plot_Bode" then converts the numbers to a graphical display. The same procedure applies with regard to the supplemental graph options menu as with the Nyquist routine.

E. ROOT LOCUS

The root locus procedure implemented in this routine is a "single parameter", or "gain locus". It graphically represents the movement of the roots of the system's closed-loop characteristic equation on the S-plane while varying the system gain. Since the location of the

characteristic roots is an indication of the system response, the designer can examine the effect of increasing gain on system behavior.

The algorithm is straightforward as can be seen in Figure 3-15. The user selects the range of variable gain values to use in the computation and the display window size. The program then plots the roots of the open-loop G-equivalent block as zeros and begins to iterate through the range of variable gains while computing the roots of the unity-feedback, closed-loop system. These roots are then plotted.

Given an open-loop equivalent transfer function:

$$G_{o.l.}(s) = K \frac{P(s)}{Q(s)}$$ where P(s), Q(s) are polynomials.

and

$$G_{c.l.}(s) = KK_v \frac{P(s)}{Q(s)+K*P(s)}$$ where Kv is the variable gain.

then

```
PLOT {open loop} Zeros.

WHILE Kv < Kmax THEN DO
  SET ClosedLoopPoly = DenCoeff + [K * Kv * NumCoeff].
  CALL RootFinder(ClosedLoopPoly).
  PLOT Poles.
  INCREMENT Kv.
ENDWHILE.
```

Figure 3-15. Root Locus Algorithm

F. TWO-PARAMETER ROOT LOCUS.

The two-parameter root locus is one of the more interesting routines in the LCS-CAD package. It is useful in a number of design situations including the most simple case of designing a cascade compensator. An example is given in Figure 3-16. The system is to be compensated with a single cascade compensator with one pole and one zero. If the blocks are combined into a single equivalent block and the closed loop characteristic equation is derived, the resultant polynomial will be a function of two unknown parameters "a" and "b". These two parameters define the location of the pole and zero of the compensator and as they are varied, the response of the system is changed. If a locus of the roots is plotted, the designer can see the effects of changing the compensator pole and zero on the overall system response.

The LCS-CAD procedure works in much the same manner as in the preceding example. The program must, however, be able to parse the user's characteristic polynomial coefficient equations in order to "understand" the relations and be able to iteratively substitute in values for a and b. The simplified algorithm is outlined in Figure 3-17.

The user is requested to provide the desired limits on the parameters a and b and to decide whether to "step" a or b.

Given a system G(s) and a compensator Gc(s)

$$G(s) = \frac{100}{s(0.1s + 1)} \qquad Gc(s) = \frac{s + a}{s + b}$$

then the closed loop equivalent transfer function is

$$Geq(s) = \frac{1000(s + a)}{1000(s + a) + s(s + 10)(s + b)}$$

and the characteristic polynomial is

$$C.P. = s^3 + (10 + b)\, s^2 + (10b + 1000)\, s + 1000a$$

If "a" is to be varied from, say, 10 to 20 and "b" from 50 to 200 then, substituting the initial values into the characteristic equation yields

$$C.P. = s^3 + 60\, s^2 + 1500\, s + 10000$$

the roots of which are

$$(s + 10)(s + 25 \pm j19.37).$$

These roots are plotted and a or b is incremented and the process repeated through the range.

Figure 3-16. Two-parameter Root Locus Example

This input page is shown in Figure 3-18 for the example problem given earlier. The parameter that is stepped is varied through five discrete values within its range and for each of these values, the other parameter is incremented fifty times. This produces a "family" of locus curves. Either a or b may be stepped at the user's option.

```
    Given a system's characteristic polynomial:

    C.P. = En s^n + En-1 s^n-1 + . . . E1s + E0

where En, En-1, etc. are algebraic expressions in a and b.
Also a is to be stepped from a1 to a2 and b varied from
b1 to b2.

Then

    SET a = a1.  SET b = b1.
    SET deltaA = (a2 - a1)/5. SET deltaB = (b2 - b1)/50.
    WHILE a < a2 DO
      WHILE b < b2 DO
        FOR i = 1 TO n STEP 1 do  {for each coefficient}
            CONVERT Ei from INFIX to POLISH.
           SUBSTITUTE values for a and b.
           COMPUTE Ei.
        ENDFOR.
        CALL RootFinder.
        PLOT Roots.
      SET b = b + deltaB.
      ENDWHILE.
      SET a = a + deltaA.
    ENDWHILE.
```

Figure 3-17. Two-Parameter Root Locus Algorithm

```
            *** Parameter Selection Page ***

You will be varying the two parameters,A and B,through a
range of values you select.  You will also choose to STEP
either A or B which means the chosen parameter's range
will be divided into five (5) equal increments to plot;
the other parameter varies smoothly through its range

A-minimum:     10
A-maximum:     20

B-minimum:     50
B-maximum:     200

Step A or B? :A_

Press <F1> to change previous entry
```

Figure 3-18. Two-parameter Root Locus Parameter Input Page

Next the user must provide a window size for viewing the root locus. This is handled in much the same manner as described for the single-parameter root locus routine. Finally, the coefficient equations must be input. Figure 3-19 shows a completed page for the example third order system equation input.

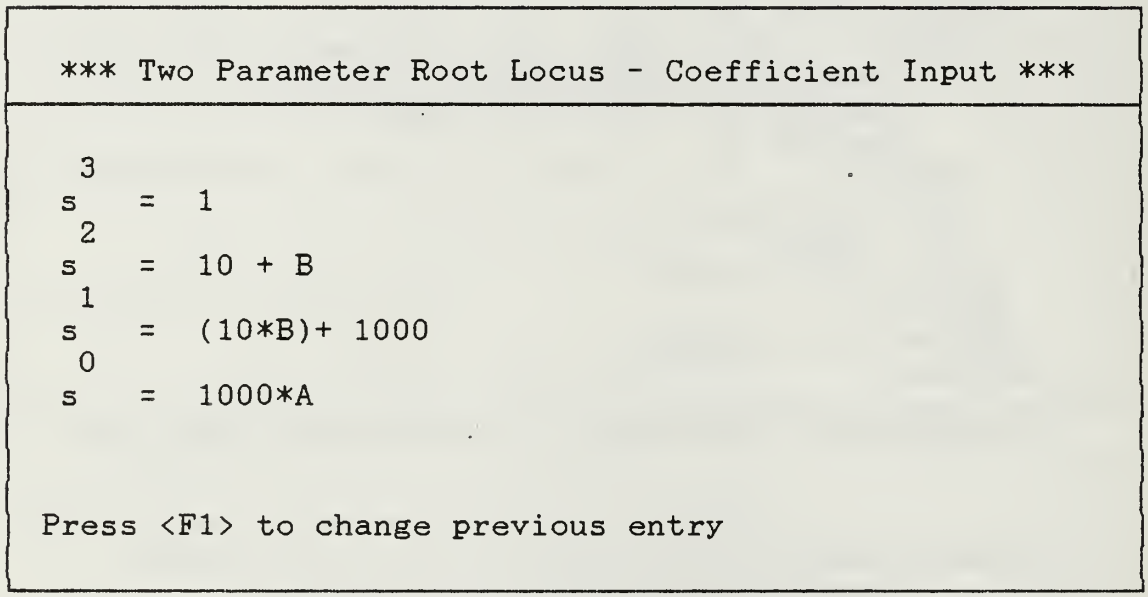


Figure 3-19. Two-parameter Root Locus Coefficient Input

Note that the equations are input in algebraic, or "infix" notation. The available operators include (+) addition, (-) subtraction, (*) multiplication, (/) division, and (^) exponentiation. These operators follow a hierarchical precedence with exponentiation operations being done first, followed by multiplication and division, and finally addition and subtraction. Operations like

multiplication and division which have the same precedence are performed from left to right when conflicts arise. To change this order of precedence, parentheses may be used around any set of operations. These parenthetical expressions have the highest priority and, when nested, the innermost operations within parentheses are done first. This scheme follows closely the protocol used in most calculators and high level programming languages.

Infix notation, while convenient for the program user, does not lend itself well to computer manipulation. A better way to represent equations for the computer is the so called "reverse Polish notation". In reverse Polish notation, the operands of an equation are entered first followed by the operator. For example, the infix expression

3 * 4 + 5

would be represented as

3 4 * 5 +

in reverse Polish notation. The numbers 3 and 4 are entered and multiplied, then 5 is entered and added to the previous result. Using the concept of a "stack" the reverse Polish expression is easy to evaluate.

Recall that a stack is a last-in-first-out queue whose operation is analogous to a stack of trays. To operate the stack, the program calls a "push" procedure to place an item on the stack, and a "pop" procedure to remove the top

item. Now, using the example given above, an arithmetic evaluation procedure can be illustrated. Figure 3-20 demonstrates such an implementation.

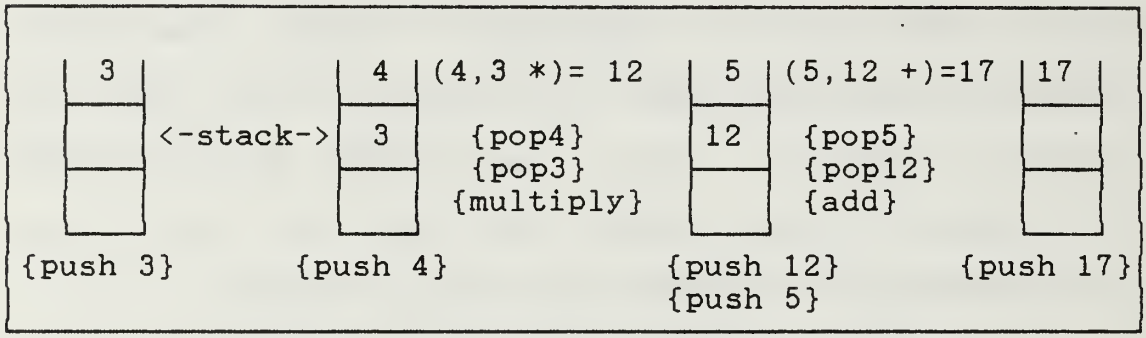


Figure 3-20. Stack Operation Example

The basic equation evaluation algorithm can be outlined then in three steps:

(1) Scan the reverse Polish equation term-by-term.

(2) If the term is a constant then push it onto the stack.

(3) If the term is an operator then pop the first two items off the stack, apply the operator, and push the result back onto the top of the stack.

When the algorithm is completed the answer to the expression will be on the top of the stack.

To get the infix equation into reverse Polish form is a bit more difficult than simply evaluating the Polish expression. Of special consideration when building the Polish form of the equation are the operator priorities and the use of parenthesis to change those priorities. A set of rules can be written which outline the conversion procedure [Ref. 2]. These rules are listed in Figure 3-21 along with an illustrative example of their application.

Given the infix expression:

6 + (5 - 4 / 2) * 3

and the following operator priority table:

Operator	Priority
^	4
*,/	3
+,-	2
operands	1
(,),spaces	0

Using a result string called RPN, and an operator stack the rules for infix to reverse Polish conversion follow:

RULE

(1) If an operand is encountered, move it to RPN.

(2) If an operator is encountered, move all higher priority operators on the stack to RPN and push the new operator onto the stack.

(3) If a left parenthesis is encountered push it onto the stack.

(4) If a right parenthesis is encountered, pop all operators off the stack and append them to RPN until a left parenthesis is encountered. Discard both parentheses.

(5) When finished with the infix expression, pop all remaining operators from the stack and append them onto RPN.

Applying these rules to the example problem above gives:

RPN	STACK	INFIX	RULE
		6+(5 - 4 / 2)*3	
6		+(5 - 4 / 2)*3	1
6	+	(5 - 4 / 2)*3	2
6	+ (	5 - 4 / 2)*3	3
6 5	+ (	- 4 / 2)*3	1
6 5	+ (-	4 / 2)*3	2
6 5 4	+ (-	/ 2)*3	1
6 5 4	+ (- /	2)*3	2
6 5 4 2	+ (- /	)*3	1
6 5 4 2 / -	+	*3	4
6 5 4 2 / -	+ *	3	2
6 5 4 2 / - 3	+ *		1
6 5 4 2 / - 3 * +			

Figure 3-21. Rules for Conversion from Infix to Reverse Polish

The LCS-CAD routine to compute and display the two parameter root locus is called "TwoParameterRootLocus". It calls several sub-programs including "Infix_to_Polish" which does the conversion of the coefficient expressions to the reverse Polish form, and "Compute_Polish" which evaluates the reverse Polish expressions. Other procedures, like "Coeff_Input", "Select_Parameter_Range", and "Select_Window_Size" , prompt the user for information required by the program.

Figure 3-22 shows the two-parameter family of curves generated by the program for the sample problem introduced at the beginning of this section. The designer wishes to place the system poles at a specific location on the S-plane, the plot will enable him to see qualitatively whether or not it is possible with a given set of parameters varying over specified ranges. A printout of the numbers used to generate the plot can then be examined for the values of a and b necessary to place the poles in the desired locations.

As with all the other graphing routines, the options menu appears after the plot is generated giving the user the opportunity to title the graph or print the graph or its corresponding numbers. An additional window appears on this graph to show a legend for easy curve identification.

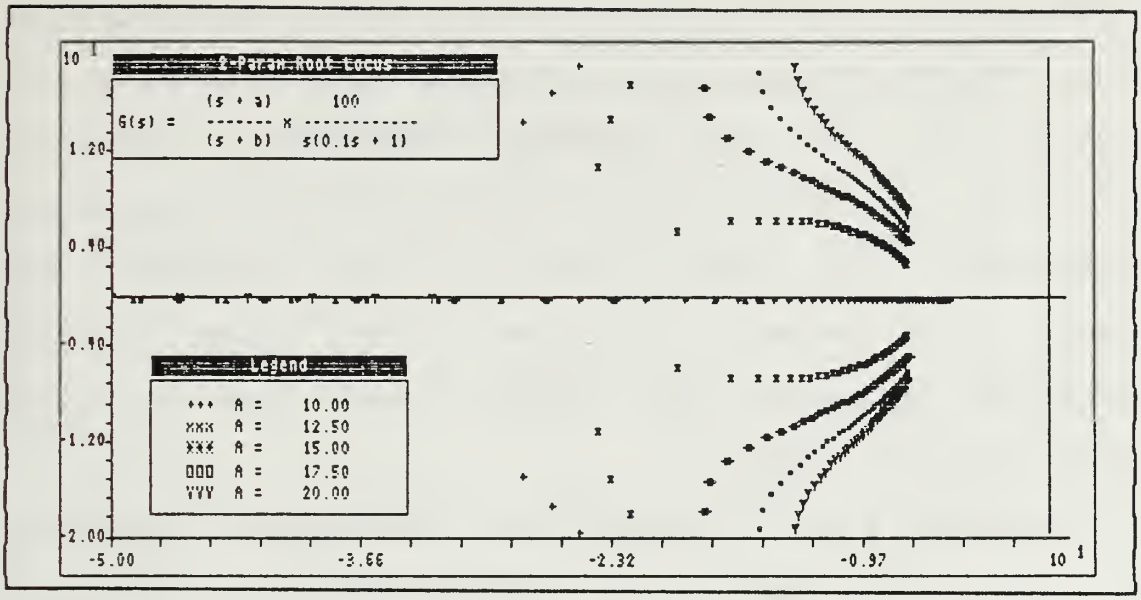


Figure 3-22. Two-parameter Root Locus Family of Curves

G. TIME RESPONSE

The time response module gives the user the capability to see how the system will respond to a "standard" input in the time domain. The user can subject his system to a choice of step, ramp, impulse, or sinusoid inputs. Since most design work will be done in the frequency domain, this module will give the designer a tie-in to the time domain and thereby a physical interpretation of the system's response.

The time response algorithm first converts the continuous-time, input-output description of the system into a discrete-time, state-space equivalent. The states of the system at any time and the output can then be generated knowing only the input and the previous system

states. This process is repeated for discrete time steps until the desired end time is reached. This method is, of course, not the only approach available. Numerical differential equation solvers, particularly the Runga-Kutta method, are often used to solve this type of problem. The "sampled-data" method was compared to the Runga-Kutta and performed favorable when using a small step size and a large number of samples.

The first step in solving the time-response problem is to convert the input-output representation into a matrix form. An example of this conversion procedure is shown in Figure 3-23. The form of the A-matrix is always the same with one's along the upper co-diagonal and the transfer function denominator coefficients along the bottom row. This is the so-called "companion form" [Ref. 3]. Likewise, the C-matrix contains the coefficients of the denominator polynomial. The single B-matrix entry indicates that the system only has one input as expected.

To convert the system to a discrete-time equivalent, the continuous-time system

$$\underline{\dot{x}}(t) = \mathbf{A}\underline{x}(t) + \underline{B}u(t)$$

$$y(t) = \underline{C}\underline{x}(t)$$

must be "mapped" into the system

$$\underline{x}(k+1) = \Phi\underline{x}(k) + \underline{\Gamma}u(k)$$

$$y(k) = \underline{C}\underline{x}(k).$$

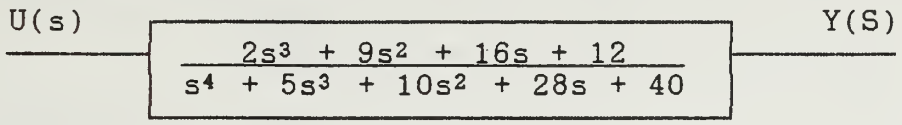


Figure 3-23(a). G(s) in Input-Output Block Form

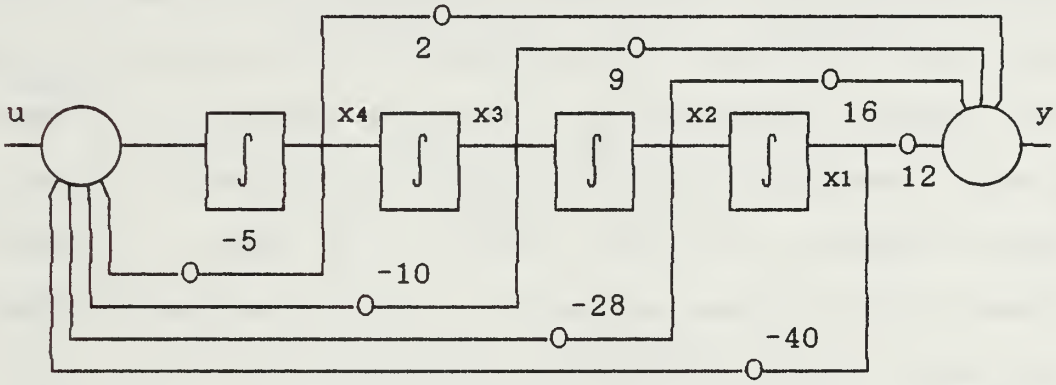


Figure 3-23(b). Flow Diagram of G(s).

$$\dot{x} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -40 & -28 & -10 & -5 \end{bmatrix}}_{A} x + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{B} u$$

$$y = \underbrace{\begin{bmatrix} 12 & 16 & 9 & 2 \end{bmatrix}}_{C} x$$

Figure 3-23(c). State-Space Representation.

Figure 3-23. Conversion of Input-Output Block Diagram to State-Space Matrix Form

Knowing the relationship between the two systems, it can be shown that

$$\Phi = I + A\,\mu(T)$$

and $\Gamma = \mu(T)B$

where T is the sampling period and μ is defined as

$$\mu(T) = \int_0^t e^{A\sigma}\, d\sigma = T \sum_{k=0}^{\infty} \frac{A^k T^k}{(k+1)!}$$

This infinite series expansion for μ can be approximated by taking enough terms to ensure precision to some acceptable value. Once the μ matrix has been computed, then the Φ and Γ matrices can also be calculated. Assuming zero initial conditions for all the states at time t=0 gives a starting point from which the states at t = 0 + t may be computed, and so on until t = t_{max}. Since the output of the system is simply one of the states, it can be extracted from the computations and plotted.

The LCS-CAD procedure which implements the time response module is called "TimeResp". The algorithm is outlined in Figure 3-24. The actual program implementation appears more complex because of the mechanics of performing the matrix and vector operations. This is simplified somewhat by the procedures "Matrix_Multiply" which multiplies two square matrices together, "Matrix_Vector_Mult" to multiply a matrix and a vector together, and "Scalar_Mult" which multiplies each of the elements of a matrix by a scalar constant.

The time increment is set to a constant value of 0.0005 seconds giving 2000 calculation increments. Because of the time required to plot all these increments, only one in five points are actually graphed.

```
Given a closed-loop transfer function of the form:
     G(s) = K  an s^n + an-1 s^n-1 + ... + a1 s + a0
               bm s^m + bm-1 s^m-1 + ... + b1 s + b0

 where m ≥ n and K is block gain.  If T is to be
 the sampling time, then

            *** Fill the A-matrix ***
FOR i,j = 1 TO m DO
    IF j = i+1 THEN SET A[i,j] = 1.{1's in upper
               ELSE SET A[i,j] = 0.  co-diagonal}
    ENDIF.
ENDFOR.
FOR i = 1 TO m DO
    A[m,i] = -bi .  {denom. coeff along bottom row}
ENDFOR.
            *** Fill the C-matrix ***
 FOR i = 1 TO n DO
    SET C[i] = ai  * K.
 ENDFOR.
        *** Initialize Psi and Atemp ***
 SET Atemp = A.
 SET Psi = I + (A * T/2).{this is first term in series}
    *** Compute additional series terms ***
 WHILE  Maxrowsum >0.1% DO {check contribution of next
                            term in series}
   SET k = 2.
   SET Psi = Psi + (A^k * T^k)/(k + 1)!.
 ENDWHILE.
 SET Psi = Psi * T.
      *** Calculate the Phi Matrix ***
  SET Phi = I + (A * Psi).
      *** Calculate the Gamma Vector ***
  SET Gamma = Psi[i,m] {since input vector U is
                        [0 0 ... 0 1]^T, Gamma is last
                        column of Psi matrix}
      *** Compute the next state vector ***
  SET xold = 0. SET t = 0.
  WHILE t < tmax DO
    SET xnext = Phi * xold + Gamma * input.
    SET y = C * xold.
    PLOT y.
    SET xold = xnext.
    SET t = t +  t.
  ENDWHILE.
```

Figure 3-24. Time Response Module Algorithm

H. UTILITIES

Several utility procedures are available from within the LCS-CAD program. These enable the user to display the current loop blocks in either polynomial or factored form, or to enter an arbitrary polynomial and have the program find and display the roots. There are three procedures which implement these functions. The first is "ShowFactors" which, as the name implies, allows the user to view the loop blocks in factored form. The procedure is essentially an on-screen formatting routine since the factors of every block have already been calculated and stored in the block record.

The second procedure is "ShowPoly". It is the complementary routine to ShowFactors. It formats the polynomial form of the loop's block transfer functions on-screen for viewing. Here too, no numeric calculation is required as the block polynomial coefficients are available in the block record.

The third procedure is called "UserPoly". This routine allows the user to input the coefficients of a polynomial and then calls the subprogram "RootFinder" (discussed earlier) to find the roots of the polynomial. The program then displays the factors in much the same manner as the ShowFactors routine.

I. HELP SCREENS

LCS-CAD has on-line help available from any of the three menus. The help screens are invoked from menus which resemble the program menu. From the help menus the user may select any of the choices normally available to him from the program menus and help screens will be displayed with help information specific to the function selected.

From the Main (program) Menu, the user may select <H>-HELP. This will bring up the Main (help) Menu. If the user select help with the <I> Input/Change function, a second help menu is displayed much as it is with the main menu selections. From there, the user may select any of the input menu choices and one or more help screens for that choice will be displayed. A similar secondary help menu is displayed if the <U> Utilities help is requested.

Unlike the Main Menu help, which can get help for any of the other menu items, if the user is in the Input Menu or the Utilities menu and calls for help from there, only the Input Help Menu or Utilities Help Menu respectively will be displayed. Figure 3-25 shows the relationship of the help screens to the program menus.

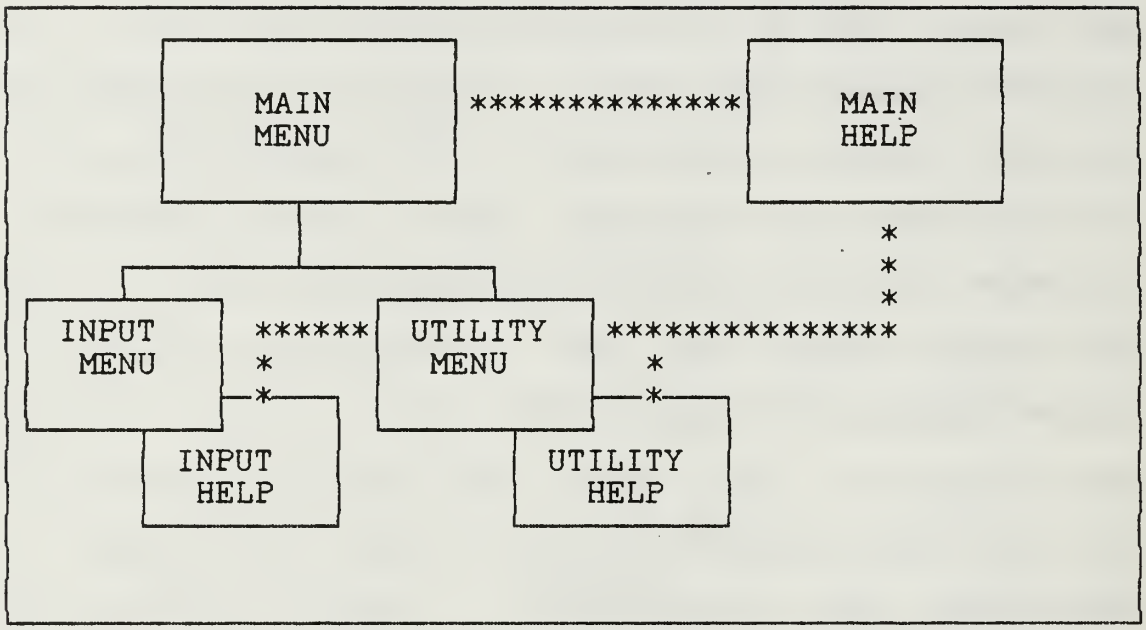


Figure 3-25. Help Available within LCS-CAD

IV. SUMMARY AND RECOMMENDATIONS

The LCS-CAD program is a useful and powerful tool for the designer and a simple and easy to use package for the student. The program supports most of the major design and analysis tools required for continuous-time, linear control system work including Bode, Nyquist, single- and two-parameter root locus, time response, and automatic transfer function block manipulation.

The program does, however, have its shortcomings and areas where improvements are possible. Several of these areas are discussed below.

(1) Currently, the program's size is constrained by the limits of the Turbo Pascal compiler. The routines could be made to run faster and the code size streamlined if the code were translated into another language which supported memory usage beyond the 64K limit imposed by Turbo Pascal. This would eliminate the need for chaining program segments into and out of memory to disk. This project should not be undertaken lightly, since Turbo Pascal is unique in its support for low-level system calls and in-line assembly language code, both of which are used in LCS-CAD routines. The best potential alternative choice for a language now appears to be Modula-2. It is similar to Pascal in construct, but allows full memory use and at least one such compiler now available (LogiTech Modula-2) appears to support most Turbo Pascal functions.

(2) Additional modules could be added to the software. A Parameter Plane module, for instance, could be added to enhance the value of the package. This is yet another tool which is a bit more sophisticated and can be applied to more difficult design problems than the tools currently in the LCS-CAD system. Another possible candidate module for inclusion in the package is the "Function Minimization"

subroutine. This routine allows the user to select a "cost function" and the program will vary the parameters of the problem to minimize the cost function. This is a type of optimal control method.

(3) Several improvements to existing routines could be made. These are listed below in no particular order or significance.

a. Adding a moveable cursor to the graphics screens to allow the user to "point" to an area of interest on a curve and have the important parameters printed on-screen. For instance, in the Bode routine, the user could move a cursor to the gain crossover point and have the frequency and the phase margin printed.

b. Adding the ability to handle the state-space form of input. Currently, only the input-output description of a system is allowed, although the state-space form is used extensively in the time response module for computation. This would add even greater flexibility to the user's choice of input.

c. Additional error trapping routines need to be added to the program. Fairly extensive efforts have been applied to keep user inputs from "bombing" the program, however, when an error occurs and aborts the program, it can be most frustrating to a user. DOS-level interrupt handlers can be written to take care of most common error like divide-by-zero and I/O faults. These programmed interrupt handlers can override the default DOS handlers which issue a cryptic message and abort to the operating system.

d. A "zoom" feature is needed on the graphics routines which will allow the user to change the plotting scales without rerunning the entire graphics routine again. This may be accomplished with strategically placed IF/THEN statements.

e. A more sophisticated input routine may be devised to allow the user to input the ENTIRE block diagram with multiple loops. The difficulty of implementing this option is to maintain a user interface which is simple and intuitive. Block interconnections and feedback path descriptions can easily become confusing and ambiguous.

f. Add the capability of the program to handle discrete time systems described by Z-domain functions.

These suggestions for future improvements to the LCS-CAD program represent specific changes as well as areas for broader, more in-depth work. The modular nature of the program structure and its menu-driven format make it a simple matter to append additional subroutines. The Pascal language with its inherent top-down structure makes for a more readable program and enables follow-on programmers to quickly and easily understand the algorithm implementation and alter it to suit their needs.

LCS-CAD is a full-featured engineering program written to operate on the most popular desk-top computer available today. With periodic maintenance and updates, this program can meet the needs of control system designers and student for many years to come.

APPENDIX A. INPUT UTILITIES

Appendix A is an excerpt from the user's documentation for a set of Turbo Pascal input and menuing utilities. These utilities, called TURBO-UT.PAS were written by:

Donald R. Ramsey

Larry Romero

727 Bunker Hill #70

Houston, Texas 77024

and distributed through the public domain. The utilites consist of a number of procedures and functions which allow the programmer to easily write sophisticated menuing systems and input routines. These utilities were used extensively in building LCS-CAD.

The following documentation describes how the programmer should use the procedures and function in a program including calling methods and parameter constraints.

DOCUMENTATION FORM:

The procedures will be given in the following form:

```
Procedure name(Variable 1,     { type var: description of var }
                  .                          .
                  .                          .
               Variable n)     { type var: description of var }
   Description of routine and calling method.
   Discussion if necessary.
   What is returned to the main program.
```

PROCEDURES:

```
Msg( String        { str: string to be displayed  }
     Col,Row)      { int: Column & row for display}
   Description: Display a message at the specified column
     and row of the video.
```

```
Center( String,          { str: string.to be displayed  }
        Col,Row,         { int: Col & row to center on  }
        Line length)     { int: length of line to center on }
   Description: clear the line to center on from the col,row
     to the line-length, then Center the string on this line.


InvVideo( String )     { str: String to display in inverse video }
   Description: display a string in inverse video and return
     to the calling routine with the background set to black
     and the text white.


Color(BackGnd,      { int: the desired background color }
      Txt )         { int: the color for the text       }
   Description: color the video as desired.


Box( X1,Y1,   { int: upper left col & row for frame }
     X2,Y2.   { int: lower right col & row          }
     divider){ int: the row position for a dividing line}
   Description: draw a box that has an optional line as a
     divider. If you do not wish to have the extra line
     divide the box, just set Z = Y1.


Option;
   Description: allow the user to press a key and return
     that character as an upper character to the calling
     routine.
   Returned : Ch ( upper case )


StripSpaces( String,    { str: string to strip spaces from end }
             NewStr )   { str: variable to return }
  Description: Strip spaces from the end of a string.
  Returned   : NewStr is a variable parameter that is returned to
               to the calling routine.
```

```
ClrWnd( X1,Y1,  { int: upper left of area to clear }
        X2,Y2)  { int: lower right of area to clear }
   Description: this is an alternate method to that of
     defining a window and doing a CLRSCR for that window.
     The advantage of this method is that the original
     window is  left intact and operations can be performed without
   keeping track of the original window. The disadvantage is
       that it is a little slower than CLRSCR.


SaveScreen
  Description: save a video display in memory for a quick
     flash back when necessary.


FlashScreen
  Description: this routine will re-display the screen saved
     by the SaveScreen procedure.


Beep( Tone,     { int: the tone to sound (Ex: 350 ) }
      Duration){ int: the time to delay (Ex: 500 ) }
  Description: turns on a tone for the desired duration
     then terminates the sound. You may test the sounds
     by running TSTSOUND.COM.


Say_cap_num
  Description: display on the 25th line of the screen
     the current status of the CAP, NUM, Ins keys.


Set_cap_num( Caps,     { ch: set to 'C' for caps }
             Num,      { ch: set to 'N' for nums }
             Ins)      { ch: set to 'I' for insert }
  Description: set the keypad as desired by the programmer.
     Ex: Set_cap_num('C','N','I')
           will set the cap lock, the num lock, and turn
           insert on.
     Ex: Set_cap_num(' ','N',' ')
           will set only the num lock.
```

```
Ck_edit_key( Ch )   { ch: variable returned }
  Description: A routine to determine if an edit key (Home, End,
     arrow key, etc.) or a function key was pressed.

  Discussion : IBM returns a two byte string for any edit
     or function key that is pressed. In order to test for this
     2 byte string you must read the keyboard twice, testing
     the second byte for the edit or function key value. So,
     you must read the KBD for a character and if a key is
     pressed, call the routine (the procedure does the second
     READ(kbd,ch)). At present, only the edit keys and the
     10 function keys are checked. However, you could easily
     expand the procedure to check the shift states, etc.
     Example call:
       READ(kbd,ch);
       if keypressed Ck_edit_key(Ch);

  Returned    : the procedure will modify the variable
     Ch if one of the edit or function keys was pressed or
     leave Ch as it was in the first READ(kbd,ch) if no edit or
     function key was pressed. You should look at the routine
     to determine what Ch will return if it has been modified
     (I always have to look, but, I know the function keys
      will return ^a for <F1>,...,^i for <F9>,^j for <F10>).



Get_template( template_num  { int: the number of the template  }
              template )    { str: variable returned to caller }
  Description: get the required template for use in the
     Input procedure.



Input( type,     { ch: 'A' for alphanumerics
                       'N' for numbers
                       'F' for formatted entries }
       default,  { str: the default string that will be displayed
                        in the input field }
       col,row,  { int: the position for the input }
       length,   { int: the field length for the input or
                        the template number if a formatted entry }
       uppercase { boo: true for uppercase letters
                        false otherwise }
       F1        { boo: variable returned true if the <F1> key
                        was pressed for the entry }
       F10 )     { boo: variable returned true if the <F10> key
                        was pressed for the entry }
```

Description: This is the main procedure for getting input from a user. The numeric keypad, the Ins key, and the CAP lock keys are set as desired by the programmer and these keys are constantly monitored to see if their status has been changed by the user. The key status is displayed on the 25th line of the screen in inverse video. As presently configured, the status of these three keys can be changed at any time by the user.

The routine allows the user to use the keyboard as he would normally expect. The arrow keys function when the NUM lock is not set, the Home and End keys respond to send the cusor to the start or end of the input line, and the Ins key state will allow a letter to be inserted in the input string if it is set to ON.

Returned : the entry of the user is return to the calling procedure in the global variable ANSWER.

```
Prompt( string1,    { str: string to be displayed on line 22 }
        string2 )   { str: string to be displayed on line 23 }
```

Description: clear line 22 and 23 of the video then display string1 and string2 there.

```
Say_prompt( prompt_num )  { int: the prompt to display }
```

Description: specify string1 and string2 to be dispalyed in the Prompt procedure. This is used primarily by the Input_handler.

```
Get_default( Default_num,  { int: the default for an input }
             Default )     { str: the variable returned    }
```

Description: provide the defaults to display in the input field for the Input_handler.

```
Do_validation( Valid_num,  { int: the number of the
                                  validation routine }
               Valid )     { boo: variable returned false if
                                  invalid entry }
```

Description: provide a routine to validate any entries that were made from the Input_handler. If the entry is invalid on return from this procedure, the Input_handler will require the user to re-enter the data.

Returned : the boolean variable VALID is returned.

```
Input_handler( string, { str: 5 character string (Ex: 'N0108 )
                                1st ch:   N  for new entries
                                          C  for changes
                                          D  for re-display
                                nums 1-2: first element number of
                                          P[] array to use
                                nums 3-4: last element of the
                                          P[] array to use )
                     Escape ) { boo: variable returned true if <F1> was
                                pressed at the first entry }
```

Description: this handler provides for full screen editing of user inputs, provides a means for changing entries, and redisplays a record.

Discussion : the programmer must provide the P[] array, specifying each input that is required. The form for P[] is

```
     P[1] := '2505A02501T010102'
       .
       .
     P[n]  where n is limited to 35 (35 entries per page)
```

Each element of P[] is defined below:

element No.	Description
1-2	the column for input
3-4	the row for input
5	type input- A for alphanumerics N for numbers F for foramtted entries $ for dollar entries
6-8	the length of the input field or the template number if a formatted entry.
9-10	array element of Filvar[] array. Filvar[] is the global variable that is returned to the calling routine.
11	set to T if you wish the Caps lock set
12-13	the default number (see Get_default)
14-15	the prompt number (see Say_prompt)
16-17	the validation No.(see Do_validation)

The programmer must also provide for defaults, prompts, and validation. The placement of these procedures is provided in the utility.

Example: the following procedure will call the handler:

```
procedure Get_inputs;
  procedure Get_variables;
   begin
     P[1] := '2505N00801F010201';
     P[2] := '2607A02502T020203';
   end;
 begin
   Get_variables;
   Input_handler('N0102');
 end;
```

This example will provide input for two variables. On return from the handler, FILVAR[i] will contain the input from the parameters specified in P[1] and FILVAR[2] will contain the input from P[2].

Changes can be made to Filvar[1..n] by calling the handler in the change mode. For example, if there were 7 variables in the Filvar array, the call Input_handler('C0207') would allow changes to Filvar[2] thru Filvar[7] (note that Filvar[1] was skipped by this call).

The handler may also be called in the Display mode. It will then display all variables except the numbers. The numbers may be displayed formatted using the FMT_REAL function.

Returned : global variables Filvar[1..n] (max n = 35)
variable ESCAPE will return true if
<F1> was pressed at the 1st field of input.

MainMenu

Description: provide a skeleton for a main menu. The procedure will draw a box around the menu items and verify the choice of the user. All that is required of the programmer, is the menu selections and a list of OKchoices (Okchoices is a list of all the choices that may be selected by the user)

FUNCTIONS

```
Fmt_Real(number,    { real: number to format }
         lenght,    { int:  the total length of the digits, commas,
                            and the decimals }
         decimals)  { int:  the number of decimal places }
```

Description: a function to format a real number with a commas
and the decimals as desired.
Example: Fmt_Real(1010.258,7,2) would return 1,010.25

UpcaseStr(S) { str: string to convert to upper case }
Description: to convert any string to upper case characters.
This function may be useful when using the Turbo Toolbox for
converting an index string to all upper case letters.

ConstStr(Character, { the cahracter to fill the string with }
Number) { the number of characters in the string }
Description: fill a string with the character of the programmers
choice. This would be useful for drawing a line of characters on
the screen.
Example: gotoXY(1,4); write(ConstStr('=',80));
This example would draw a line of equal signs at line 4 of
the video.

APPENDIX B

Appendix B is a listing of the Pascal source code which makes up the major modules of the LCS-CAD program. In general, driver programs are included first followed by their supporting subprograms. Not included in the listings are the source modules for the Borland International Turbo Graphix Toolbox and the input routines described in the thesis text and in Appendix A.

```
1                   {*****************************************************************}
2                   {** Program Control_CAD contains the calls for the main menu proc **}
3                   {** and also the files for the single-parameter root locus program**}
4                   {*****************************************************************}
5
6
7                   program Control_CAD;
8                   {*****************************************************************}
9                   {**  The following include-files contain routines and procedures  **}
10                  {**  to handle graphics calls from the main procedures of this    **}
11                  {**  program.  They are a part of Borland International's Turbo   **}
12                  {**  Graphix Toolbox, a commercially available product.           **}
13                  {*****************************************************************}
14
15                  {$I typedef.sys}
16                  {$I graphix.sys}
17                  {$I kernel.sys}
18                  {$I windows.sys}
19                  {$I polygon.hgh}
20                  {$I axis.hgh}
21
22                  {*****************************************************************}
23                  {**  These include files are utility procedures and functions to  **}
24                  {**  facilitate input handling.  They are public domain procedures**}
25                  {*****************************************************************}
26
27
28                  {$I UT-MOD01.INC }
29                  {$I UT-MOD02.INC }
30                  {$I UT-MOD03.INC }
31
32                  {*****************************************************************}
33                  {** GrapMenu contains procedures to open a window on each graphic **}
34                  {** screen with a menu of user options.  The selections allow     **}
35                  {** printing a hardcopy of the graph on a dot-matrix printer,     **}
36                  {** dumping the data points to the printer, or constructing and   **}
37                  {** placing a title block on the graph.                          **}
38                  {*****************************************************************}
39
40                  {$I GrapMenu.inc}
41
42
43
44                  {*****************************************************************}
45                  {**  The include-file Roots.Inc is a set of procedures to find    **}
46                  {**  the roots of a given polynomial in coefficient form.  The    **}
47                  {**  algorithm use is a modified version of the Bairstow method.  **}
48                  {*****************************************************************}
49
50                  {$I Roots.INC }
51
52
53
54                  {*****************************************************************}
55                  {**  The Include-file SHOWROOT.inc will display the input or      **}
56                  {**  calculated roots of the G-equivalent block diagram. Both     **}
```

```
                        {**  poles and zeros will be displayed.                    **)
                        {**********************************************************)

                        {$I ShowRoot.inc}



                        {**********************************************************)
                        {**  The include-file ROOTLOC.INC is a procedure to draw the  **)
                        {**  plot of a single-parameter root locus through a user-suppled **)
                        {**  range of gain (K).                                    **)
                        {**********************************************************)

                        {$I RootLoc.inc}



                        Procedure MainMenu;
                         var
                           I,Tab              : integer;
                           Okchoices          : set of char;
                           Helpfile           : file;

                         procedure ProgramExit;  (displays warning about program about to end)
                        B------------------------begin
                        |                          Clrscr; Center('This Program is about to end',1,11,80);
                        |                          highvideo; Center('Verify Ok (Y/N)',1,13,80); lowvideo;
                        |  B-----------------------repeat
                        |  |                         Option; if not (Ch in ['Y','N']) then beep(350,150);
                        |  E-----------------------until Ch in ['Y','N'];
                        E---------------------------end;


                         procedure MenuItem(pick:char;description:str80;color:integer);
                              {allows easyselection of menu colors)
                        B------------------------begin
                        |                          textcolor(color);
                        |                          write(' ':Tab,'<'); textcolor(White); write(pick);
                        |                          textcolor(color); writeln('> ',description);
                        E------------------------end;


                        B------------------------begin {MainMenu)
                        |                          ClrScr; TextColor(White);
                        |                          GotoXY(1,24); Write('Revision date: 9/04/86');
                        |
                        |                          Center('*** MAIN  MENU ***  ',1,4,80); {display main menu)
                        |                          for I:= 1 to 4 do  writeln('');
                        |                          Tab:= 23;
                        |
                        |                          MenuItem('1','Input/Change Transfer Function(s)',green);
                        |                          writeln;
                        |                          MenuItem('L','Location of Char. Eq. Roots',green);
                        |                          MenuItem('F','Frequency Analysis',green);
                        |                          MenuItem('R','Root Locus Analysis',green);
                        |                          writeln;
                        |                          MenuItem('P','Two Parameter Root Locus',green);
```

```
114 |                                Menuitem('T','Time Response',green);
115 |                                Menuitem('U','Utilities',green);
116 |                                writeln;
117 |                                Menuitem('H','Help',blue);
118 |                                Menuitem('Q','Exit Program',lightmagenta);
119 |                                HighVideo;
120 |                                TextColor(Yellow);
121 |                                Box(20,2,65,23,6);writeln('');
122 |                                Set_Cap_Num('C',' ',' ');Say_Cap_Num;
123 |                               TextColor(White); Center('Press Your Selection',21,22,38); LowVideo;
124 |                                        (sets legal choices depending on whether user has entered
125 |                                         block information or not)
126 |                               if not(NBlocks in [1..9]) then OKchoices := ['I','U','H','Q','P']
127 |                                                         else OKchoices := ['I','L','F','R','U','T','H','Q','P'];
128 |  B-------------------------repeat
129 |  |                              Option; if not (Ch in OKchoices) then
130 |  |  B-----------------------------------begin
131 |  |  |                                     Beep(350,150); TextColor(white);
132 |  |  |                                     if NBlocks <= 0 then
133 |  |  |  B--------------------------------------begin
134 |  |  |  |                                        msg('WARNING : First INPUT the block descriptions!',1,25);
135 |  |  |  E--------------------------------------end;
136 |  |  E-----------------------------------end;
137 |  E-------------------------until Ch in OKchoices;
138 |  B-------------------------case Ch of
139 |  |  B-----------------------'I' : Begin
140 |  |  |                              Assign(InputFile,'Input.chn');
141 |  |  |                              Chain(InputFile);
142 |  |  E-----------------------------end;
143 |  |                            'R' : Root_Locus(G_eq) ;
144 |  |  B-----------------------'P' : Begin
145 |  |  |                              Assign(TwoParamFile,'TwoParam.chn');
146 |  |  |                              Chain(TwoParamFile);
147 |  |  E-----------------------------end;
148 |  |                            'L' : ShowRoots(G_eq) ;
149 |  |  B-----------------------'F' : Begin
150 |  |  |                              Assign(FreqFile,'Freq.chn');
151 |  |  |                              Chain(FreqFile);
152 |  |  E-----------------------------end;
153 |  |  B-----------------------'T' : Begin
154 |  |  |                              Assign(TimeFile,'TimeResp.chn');
155 |  |  |                              Chain(TimeFile);
156 |  |  E-----------------------------end;
157 |  |  B-----------------------'U' : Begin
158 |  |  |                              Assign(Utilfile,'UtilMenu.chn');
159 |  |  |                              Chain(Utilfile);
160 |  |  E-----------------------------end;
161 |  |  B-----------------------'H' :  Begin
162 |  |  |                               Assign(Helpfile,'HelpMenu.chn') ;
163 |  |  |                               Chain(Helpfile);
164 |  |  E-----------------------------end;
165 |  |  B-----------------------'Q' :  begin
166 |  |  |                                ProgramExit; if Ch='Y' then Exit := true;
167 |  |  E-----------------------------end;
168 |  E-------------------------end;
169 E----------------------------end;
170
```

```
171                          {*********************************************************}
172                          {*             Program Starts Execution                  *}
173                          {*********************************************************}
174
175 B------------------------begin
176 ¦                          InitGraphic;LeaveGraphic;ClrScr; {initialize screen}
177 ¦
178 ¦                          Fillchar(s,100,#205); S:= copy(S,1,80); {use special character to draw line}
179 ¦
180 ¦                          Exit:=false;   {initialize boolean}
181 ¦
182 ¦ B----------------------repeat
183 ¦ ¦                         MainMenu;   {repeatedly call main menu until user wants to quit}
184 ¦ E----------------------until Exit = true;
185 ¦                          Set_Cap_Num(' ',' ',' ');Say_Cap_Num;  {set caps, insert, and num lock off}
186 ¦                          NBlocks := 0;  {reset problem}
187 E------------------------end.
```

```
 1                       Program Input(input,output);
 2
 3                       {$I TYPEDEF.SYS}       {program type & variable definitions - same as CAD.PAS}
 4                       {$I DIRECTRY.INC }     {displays directory of available blocks}
 5                       {$I UT-MOD01.INC }     {input utility routines}
 6                       {$I UT-MOD02.INC }
 7                       {$I UT-MOD03.INC }
 8                       {$I EXPAND.INC }       {expands factors into polynomials}
 9                       {$I ROOTS.INC  }       {finds factors (roots) of polynomials}
10                       {$I MAKEGEQ.INC}       {combines loop blocks into single equivalent block}
11                       {$I INPUTHLP.INC}      {help screen driver}
12
13
14
15
16                       {*************************************************************************}
17                       {** Block input is called from input, add, and change procedures to input  **}
18                       {** a single block.                                                       **}
19                       {*************************************************************************}
20
21                       Procedure Block_Input(var Block_Description:Blocks;BlockIndex:integer;readerror:boolean);
22
23                         var
24                           i,j,code     : integer;     {i,j: loop counters; code: error code from Var proc}
25                           temp,temp2   : char;        {temporary holding var for input}
26                           VertPos,
27                           HorizPos,
28                           PosCounter   : integer;     {screen prompt positioning variables}
29                           Exponent     : string[2];   {exponent of S for coeff input form display}
30                           Specification: string[5];   {Input_Handler calling string}
31                           Strg         : string[2];
32                           Filename     : str20;       {file name for storing block data}
33                           ReadFile     : file of Blocks; {file for reading in block filenames}
34                           NZeros_old,
35                           NPoles_old   : integer;
36                           test         : real;     {temporary variable to hold results of "val" }
37                                                    {until conversion is validated}
38
39
40
41
42
43                           {*************************************************************************}
44                           {**    Begin processing Factored form Input-Internal routine           **}
45                           {*************************************************************************}
46
47
48                         Procedure Input_Factored (Zeros_or_Poles:Str5;NFactors:integer;var RealPart,
49                                                   ImagPart: PolyArray);
50                         var
51                          i,j    : integer;
52                          test   : real;      {holds "val" results until conversion is validated}
53
54 B-----------------------begin
55 |
56 |                         P[11] := '0905N01011-000101';  {set up input handler control strings}
```

```
57  ¦                          P[12] := '2505N01012-000101';  {for factored form input -- odd strings)
58  ¦                          P[13] := '0907N01013-000101';  {for real parts, even strings for imag)
59  ¦                          P[14] := '2507N01014-000101';
60  ¦                          P[15] := '0909N01015-000101';
61  ¦                          P[16] := '2509N01016-000101';
62  ¦                          P[17] := '0911N01017-000101';
63  ¦                          P[18] := '2511N01018-000101';
64  ¦                          P[19] := '0913N01019-000101';
65  ¦                          P[20] := '2513N01020-000101';
66  ¦                          P[21] := '0915N01021-000101';
67  ¦                          P[22] := '2515N01022-000101';
68  ¦                          P[23] := '0917N01023-000101';
69  ¦                          P[24] := '2517N01024-000101';
70  ¦                          P[25] := '0919N01025-000101';
71  ¦                          P[26] := '2519N01026-000101';
72  ¦                          P[27] := '0921N01027-000101';
73  ¦                          P[28] := '2521N01028-000101';
74  ¦                          P[29] := '0923N01029-000101';
75  ¦                          P[30] := '2523N01030-000101';
76  ¦
77  ¦
78  ¦                          ClrScr; TextColor(White);   {write screen titles)
79  ¦                          Center('***Block Transfer Function Input***',1,1,80);
80  ¦                          HighVideo;
81  ¦                          writeln;writeln(s);TextColor(green);
82  ¦                          if Zeros_or_Poles = 'ZEROS' then
83  ¦                              writeln('NUMERATOR Transfer Function Input -- FACTORED Form')
84  ¦                          else
85  ¦                              writeln('DENOMINATOR Transfer Function Input -- FACTORED Form');
86  ¦
87  ¦                          HighVideo;
88  ¦                          for j:=1 to NFactors do   {type prompt strings)
89  ¦  B-----------------------------begin
90  ¦  ¦                           writeln('   s =             +j');
91  ¦  ¦                           writeln;
92  ¦  E-----------------------------end;
93  ¦
94  ¦                          str((NFactors*2+10):2,strg);        {select proper input handler)
95  ¦                                                              {control strings based on NFactors)
96  ¦
97  ¦                          if Change then specification := concat('C11',strg)  {builds string to call)
98  ¦                                    else specification := concat('N11',strg); {input handler)
99  ¦
100 ¦                          Input_handler(specification, escape);  {call the input handler)
101 ¦
102 ¦                          for j:= 1 to NFactors do                {compute the zero values from)
103 ¦  B-----------------------------begin                           {input handler string Filvar)
104 ¦  ¦                           val(filvar[2*j+9 ],test,code);
105 ¦  ¦                           if code = 0 then RealPart[j]:=test; {val conversion successful if code)
106 ¦  ¦                           val(filvar[2*j+10],test,code);      {=0, error otherwise)
107 ¦  ¦                           if code = 0 then ImagPart[j]:=test;
108 ¦  ¦
109 ¦  E-----------------------------end;
110 E-------------------------end; {procedure Input_factored)
111
112
113
```

```
114
115
116                              {*********************************************************************}
117                              {**     Begin processing Coefficient form Input-Internal procedure     **}
118                              {*********************************************************************}
119
120                            Procedure Input_Coeff(Zeros_or_Poles:Str5; NCoeff:integer;
121                                                  var Coeff: Polyarray);
122
123                           var
124                             i,j,                              {counters}
125                             NCoeff_old          : integer;  {holds old poly order if changing order}
126                             test                : real;     {holds "val" results until validated}
127
128 B-------------------------begin
129 |
130 |                           P[21]  := '0406N01021-000101';  {Input-Handler descriptors for coeff}
131 |                           P[22]  := '1B06N01022-000101';  {form input}
132 |                           P[23]  := '3206N01023-000101';
133 |                           P[24]  := '4606N01024-000101';
134 |                           P[25]  := '0408N01025-000101';
135 |                           P[26]  := '1808N01026-000101';
136 |                           P[27]  := '3208N01027-000101';
137 |                           P[28]  := '4608N01028-000101';
138 |                           P[29]  := '0410N01029-000101';
139 |                           P[30]  := '1810N01030-000101';
140 |
141 |                           NCoeff_old:= NCoeff;
142 |
143 |                           ClrScr; TextColor(White);             {print screen titles}
144 |                           Center('***Block Transfer Function Input***',1,2,80);
145 |                           writeln; write(s);
146 |                           TextColor(Green);
147 |                           if Zeros_or_Poles = 'ZEROS' then
148 |                                 writeln('NUMERATOR Transfer Function Input -- COEFFICIENT Form')
149 |                             else
150 |                                 writeln('DENOMINATOR Transfer Function Input -- COEFFICIENT Form');
151 |                           HighVideo;
152 |
153 |                              {Adjusts the coefficients if CHANGING number-zeros in block}
154 |                           if NCoeff > NCoeff_old then
155 |  B----------------------------begin
156 |  |                           for j:=1 to NCoeff_old - NCoeff do
157 |  |                              for i:=NCoeff_old + 1 downto 1 do
158 |  |                                 Filvar[i+1] := Filvar[i];  {shift the coeff down to }
159 |  |                                                           {location proper exponent}
160 |  E----------------------------end;
161 |
162 |                           if NCoeff < NCoeff_old then
163 |  B----------------------------begin
164 |  |                           for j:= 1 to NCoeff_old - NCoeff do
165 |  |                              for i:= 1 to NCoeff_old + 1 do  {shifts the coeff up to proper expon}
166 |  |                                 Filvar[i] := Filvar[i+1];
167 |  E----------------------------end;
168 |
169 |
170 |                            VertPos:=4;                    {positions prompts on screen}
```

```
171                              for i:=NCoeff+1 downto 1 do
172    B-------------------------begin
173    |                          j:=NCoeff+1 - i;
174    |                          PosCounter := (j mod 4) + 1; HorizPos := PosCounter * 14;
175    |                          If PosCounter = 1 then VertPos := VertPos + 2;
176    |
177    |                          if i<> 1 then               {prompts for coeff input}
178    |  B-----------------------begin
179    |  |                          msg('s +',HorizPos,VertPos);
180    |  |                          str(i-1:2,Exponent);
181    |  |                          msg(Exponent,HorizPos+1,VertPos-1);.
182    |  E-----------------------end;
183    E-------------------------end;
184
185                             str((20+NCoeff+1):2,strg);          {sets up and calls input handler}
186
187                             if Change then specification := concat('C21',strg)
188                                       else specification := concat('N21',strg);
189
190                             Input_handler(specification,escape);
191
192                             for j:= NCoeff+1 downto 1 do
193    B-----------------------begin
194    |                          val(Filvar[NCoeff+22-j],test,code);
195    |                          if code = 0 then Coeff[j]:=test;
196    E-----------------------end;
197  E-------------------------end;
198
199
200
201
202
203                          {*************************************************************************}
204                          {**  Begin the Block Input Routine .  This allows input of a generic block **}
205                          {**  from either input or change routines.                                **}
206                          {*************************************************************************}
207
208  B-------------------------begin {procedure Block_Input}
209  |                          with Block_Description do   {Block_Description is Blocks-type variable}
210  |  B----------------------begin                        {passed to this routine}
211  |  |                          Clrscr;
212  |  |
213  |  |                          if Change then temp2 := 'K'  {if Change option always from keyboard}
214  |  |                            else
215  |  |  B-------------------------begin
216  |  |  |                          gotoxy(1,7); textcolor(Green);write('Block No. ',BlockIndex);
217  |  |  |                          textcolor(yellow);
218  |  |  |                          msg('Will you input this block from the Keyboard (K) or Diskfile (D) ?',1,10);
219  |  |  |  B----------------------repeat
220  |  |  |  |                          Input('A','K',73,10,2,true,F1,F10);  {validate for "K" or "D"}
221  |  |  |  |                          temp2:=copy(answer,1,1);
222  |  |  |  |                          if not(temp2 in ['K','D']) then beep(350,150);
223  |  |  |  E----------------------until temp2 in ['K','D'];
224  |  |  E-------------------------end;
225  |  |
226  |  |                          if temp2 = 'D' then     { if input from diskfile, then prompt for drive}
227  |  |  B-------------------------begin                { and display directory of blocks available}
```

```
228 |  |  |                          Clrscr; HighVideo;
229 |  |  |                          Msg('***Current Data Drive is    . Press <esc> to change it!***',10,11);
230 |  |  |  B----------------------repeat
231 |  |  |  |                        input('A',copy(drive,1,1),36,11,2,true,F1,F10);
232 |  |  |  |                        ch:= copy(answer,1,1);
233 |  |  |  |                        if not(ch in ['A','B','C','D'])  then beep(350,150);
234 |  |  |  E----------------------until ch in ['A','B','C','D'];
235 |  |  |                          Drive := concat(ch,':');
236 |  |  |
237 |  |  |                          highvideo;
238 |  |  |                          Directory(drive,extension,filename,readerror);{call Directory display}
239 |  |  |                          if  not(readerror) then
240 |  |  |  B----------------------begin
241 |  |  |  |                          Assign(Readfile,filename);    {open the selected file passed back}
242 |  |  |  |                          Reset(Readfile);              {from Directory in variable Filename}
243 |  |  |  |                          Read(readfile,block_description);
244 |  |  |  |                          clrscr;
245 |  |  |  |                          msg('Is the block read from disk in the Forward (F) or FeedBack (B) path?',1,10);
246 |  |  |  |  B-------------------repeat
247 |  |  |  |  |                        Input('A','F',75,10,1,true,F1,F10);
248 |  |  |  |  |                        ch:= copy(answer,1,1);
249 |  |  |  |  |                        if not(ch in ['F','B']) then beep(350,150);
250 |  |  |  |  E-------------------until ch in ['F','B'];
251 |  |  |  |                          if ch = 'B' then FeedBack := true
252 |  |  |  |                                      else FeedBack := false;
253 |  |  |  |
254 |  |  |  E----------------------end
255 |  |  |                          else
256 |  |  |  B----------------------begin
257 |  |  |  |                          delay(1500);  {wait until directory error is displayed}
258 |  |  |  |                          window(1,1,80,25);   {expand out of directory window}
259 |  |  |  |                          clrscr;
260 |  |  |  E----------------------end;
261 |  |  E-------------------end
262 |  |
263 |  |                     else
264 |  |
265 |  |  B-------------------begin         {begin Keyboard input}
266 |  |  |                     ClrScr; TextColor(White);
267 |  |  |                     Center('Block Input Segment',1,2,80); HighVideo;
268 |  |  |                     writeln; writeln(s); writeln; TextColor(Green);
269 |  |  |                     writeln('Block ',BlockIndex,': ');
270 |  |  |                     HighVideo;
271 |  |  |
272 |  |  |                     P[1] := '6007A00201T000110';  {Fbk/Fwd path block}
273 |  |  |                     P[2] := '6011N00202-000102';  {NZeros prompt}
274 |  |  |                     P[3] := '6013N00203-000102';  {NPoles prompt}
275 |  |  |                     P[4] := '6015N00904-000101';  {Block gain prompt}
276 |  |  |                     P[5] := '6018A00205T000111';  {Fact. or Coeff. form input}
277 |  |  |
278 |  |  |
279 |  |  |                   msg('Is block in Forward (F) or Feedback (B) Path?',1,7);
280 |  |  |                   msg('What is the order of the Numerator?',10,11);
281 |  |  |                   msg('What is the order of the Denominator?',10,13);  {# poles}
282 |  |  |                   msg('What is the block gain constant?',10,15);
283 |  |  |                   if not(Change) then
284 |  |  |  B----------------begin
```

```
285 ¦ ¦ ¦ ¦                          msg('Will you enter the block in Factored (F)',10,17);
286 ¦ ¦ ¦ ¦                          msg('or Coefficient (C) form?',28,18);
287 ¦ ¦ ¦ E------------------------end;
288 ¦ ¦ ¦
289 ¦ ¦ ¦                        if Change then Input_Handler('C0104',escape)
290 ¦ ¦ ¦                                  else Input_Handler('N0105',escape);
291 ¦ ¦ ¦
292 ¦ ¦ ¦
293 ¦ ¦ ¦                        if copy(Filvar[01],1,1) = 'B' then Feedback:= true
294 ¦ ¦ ¦                                                      else FeedBack:= false;
295 ¦ ¦ ¦
296 ¦ ¦ ¦                        val(Filvar[02],NZeros,code);  {convert Nzeros to number}
297 ¦ ¦ ¦                        val(Filvar[03],NPoles,code);  {convert NPoles to number}
298 ¦ ¦ ¦                        val(Filvar[04],test  ,code);  {convert gain to number/validate}
299 ¦ ¦ ¦                        if code = 0 then K:=test;
300 ¦ ¦ ¦
301 ¦ ¦ ¦                        If copy(Filvar[05],1,1) = 'F' then Factored:= true
302 ¦ ¦ ¦                                                      else Factored:= false;
303 ¦ ¦ ¦
304 ¦ ¦ ¦
305 ¦ ¦ ¦
306 ¦ ¦ ¦
307 ¦ ¦ ¦
308 ¦ ¦ ¦                        if Factored then    {set up to call Input_Factored routine}
309 ¦ ¦ ¦ B------------------------begin
310 ¦ ¦ ¦ ¦
311 ¦ ¦ ¦ ¦                          if Change then  {if Changing an old entry, then put all old values}
312 ¦ ¦ ¦ ¦                                          {into associated Filvar[] to be displayed}
313 ¦ ¦ ¦ ¦                            for j:=1 to NZeros do
314 ¦ ¦ ¦ ¦ B----------------------------begin
315 ¦ ¦ ¦ ¦ ¦                              str(RealPartZero[j]:10:2,filvar[2*j+9]);
316 ¦ ¦ ¦ ¦ ¦                              str(ImagPartZero[j]:10:2,filvar[2*j+10]);
317 ¦ ¦ ¦ ¦ E----------------------------end;
318 ¦ ¦ ¦ ¦
319 ¦ ¦ ¦ ¦
320 ¦ ¦ ¦ ¦                          if NZeros = 0 then NumCoeff[1]:=1.0 {if order zero, then assign }
321 ¦ ¦ ¦ ¦                                  else                        {unity coefficient and go on}
322 ¦ ¦ ¦ ¦ B--------------------------------begin                        {otherwise prompt for input}
323 ¦ ¦ ¦ ¦ ¦                                  Input_Factored('ZEROS',NZeros,RealPartZero,ImagPartZero);
324 ¦ ¦ ¦ ¦ ¦
325 ¦ ¦ ¦ ¦ ¦                                  {expand factors to polynomial and store in NumCoeff array}
326 ¦ ¦ ¦ ¦ ¦                                  Expand_Poly(RealPartZero,ImagPartZero,
327 ¦ ¦ ¦ ¦ ¦                                                          NumCoeff,NZeros);
328 ¦ ¦ ¦ ¦ ¦
329 ¦ ¦ ¦ ¦ E--------------------------------end;
330 ¦ ¦ ¦ ¦
331 ¦ ¦ ¦ ¦                          if Change then        {repeat above process for Poles(denom) input}
332 ¦ ¦ ¦ ¦                            for j:=1 to NPoles do
333 ¦ ¦ ¦ ¦ B----------------------------begin
334 ¦ ¦ ¦ ¦ ¦                              str(RealPartPole[j]:10:2,filvar[2*j+9]);
335 ¦ ¦ ¦ ¦ ¦                              str(ImagPartPole[j]:10:2,filvar[2*j+10]);
336 ¦ ¦ ¦ ¦ E----------------------------end;
337 ¦ ¦ ¦ ¦
338 ¦ ¦ ¦ ¦
339 ¦ ¦ ¦ ¦                          if NPoles = 0 then DenCoeff[1]:=1.0
340 ¦ ¦ ¦ ¦                                  else
341 ¦ ¦ ¦ ¦ B--------------------------------Begin
```

```
342 |  |  |  |                                    Input_Factored('POLES',NPoles,RealPartPole,ImagPartPole);
343 |  |  |  |                                      Expand_Poly(RealPartPole,ImagPartPole,
344 |  |  |  |                                                                  DenCoeff,NPoles);
345 |  |  |  E-------------------------------end;
346 |  |  |                                LeadNumCoeff:= 1.0; LeadDenCoeff:= 1.0; {expanding factors will always}
347 |  |  |                                            {create unity leading coefficients. These variables are}
348 |  |  |                                            {used in Coeff_Input to assist in normalizing/denormalizing}
349 |  |  |                                            {coefficients for display/change and computation}
350 |  |  E-------------------------end
351 |  |
352 |  |                             else
353 |  |
354 |  |
355 |  |                               {**********Begin Coefficient form input******************}
356 |  |
357 |  |  B-------------------------begin
358 |  |  |                           if Change then   {if changing entries, restore normalized coeffs to}
359 |  |  |                                            {orig. form - doesn't confuse user. Then restore into}
360 |  |  |                                   .        {proper FilVar[] for display}
361 |  |  |                             for j:=NZeros+1 downto 1 do
362 |  |  |  B----------------------------begin
363 |  |  |  |                                NumCoeff[j] := NumCoeff[j] * LeadNumCoeff;
364 |  |  |  |                                str(NumCoeff[j]:10:2,filvar[NZeros+22-j]);
365 |  |  |  E----------------------------end;
366 |  |  |
367 |  |  |                               Input_Coeff('ZEROS',NZeros,NumCoeff); {call Input_Coeff routine}
368 |  |  |
369 |  |  |                               LeadNumCoeff:= NumCoeff[NZeros + 1]; {normalize polynomial}
370 |  |  |                               for j:= 1 to NZeros + 1 do
371 |  |  |                                  NumCoeff[j] := NumCoeff[j] / LeadNumCoeff;
372 |  |  |                               K := K * LeadNumCoeff;  {mult K by leading numerator coeff to keep}
373 |  |  |                                                       {polynomial's numeric integrity}
374 |  |  |
375 |  |  |
376 |  |  |                                RootFinder(NZeros,NumCoeff,RealPartZero,ImagPartZero,0.0,0.0);
377 |  |  |                                                       {factor for real and imag roots}
378 |  |  |
379 |  |  |                               if Change then        {repeat above process for denominator}
380 |  |  |                                 for j:=NPoles+1 downto 1 do
381 |  |  |  B----------------------------begin
382 |  |  |  |                                DenCoeff[j] := DenCoeff[j] * LeadDenCoeff;
383 |  |  |  |                                str(DenCoeff[j]:10:2,filvar[NPoles+22-j]);
384 |  |  |  E----------------------------end;
385 |  |  |
386 |  |  |                               Input_Coeff('POLES',NPoles,DenCoeff);
387 |  |  |
388 |  |  |                               LeadDenCoeff:= DenCoeff[NPoles + 1]; {normalize polynomial}
389 |  |  |                               for j:= 1 to NPoles + 1 do
390 |  |  |                                  DenCoeff[j] := DenCoeff[j] / LeadDenCoeff;
391 |  |  |                               K := K / LeadDenCoeff;  {divide K by lead Denom. coeff to keep}
392 |  |  |                                                       {polynomial's numeric integrity}
393 |  |  |
394 |  |  |                                RootFinder(NPoles,DenCoeff,RealPartPole,ImagPartPole,0.0,0.0);
395 |  |  |  E-------------------------end;{else}
396 |  |  E----------------------end; {if}
397 |  E-------------------------end; {with}
398 E----------------------end; {procedure}
```

```
399
400
401
402                          {*****************************************************************}
403                          {** This procedure prompts for the number of blocks in the loop   **}
404                          {** and iterates through calling the block input routine as       **}
405                          {** required  to allow input of all blocks for analysis.         **}
406                          {*****************************************************************}
407
408
409                           Procedure Trans_Function_Input;
410
411                          var
412                            i,j,code      : integer;
413                            InputErr      : boolean;
414                            LoopFeedBack  : boolean;
415
416  B-----------------------begin
417  ¦                         Change:= false;
418  ¦                         Clrscr;
419  ¦                         TextColor(White);
420  ¦                         Center('***Transfer Function Input Routine***',1,2,80);
421  ¦                         HighVideo;
422  ¦                         Fillchar(s,100,#205); S:= copy(S,1,80);{construct horizontal line on screen}
423  ¦                         writeln; writeln(s); writeln;
424  ¦
425  ¦                                                {begin user prompts}
426  ¦                         msg('How many blocks in this loop?',10,6);
427  ¦                         Input('N','1',43,6,2,true,F1,F10);
428  ¦                         val(Answer,NBlocks,Code);
429  ¦                         LoopFeedBack := false;
430  ¦                         for i:= 1 to NBlocks do
431  ¦  B--------------------------begin
432  ¦  ¦  B-------------------------repeat
433  ¦  ¦  ¦                           Block_Input(Block[i],i,InputErr); {call Block_input until no error}
434  ¦  ¦  E-------------------------until not(inputErr);              {and all loop blocks have been made}
435  ¦  ¦                           if Block[i].Feedback then LoopFeedBack := true;
436  ¦  ¦                           Clrscr;
437  ¦  E---------------------------end;
438  ¦                            if LoopFeedback then
439  ¦  B---------------------------begin
440  ¦  ¦                          msg('Is the Loop FeedBack Negative (N) or Positive (P)?',1,10);
441  ¦  ¦
442  ¦  ¦  B-------------------------repeat     {validate input answer}
443  ¦  ¦  ¦                          Input('A','',55,10,2,true,F1,F10);
444  ¦  ¦  ¦                          ch:=copy(answer,1,1);
445  ¦  ¦  ¦                          if not (ch in ['N','P']) then beep(350,150);
446  ¦  ¦  E-------------------------until ch in ['N','P'];
447  ¦  ¦
448  ¦  ¦                           if ch ='N' then NegFeedback := true
449  ¦  ¦                                      else NegFeedback := false;
450  ¦  E---------------------------end;
451  ¦
452  ¦                            First_run := false;
453  ¦                            Make_Geq;   {once loop finished, compute equivalent single block}
454  E-----------------------end;
455
```

```
456
457
458
459
460                          {*****************************************************************}
461                          {**  The following procedure allows the user to change the contents*}
462                          {**  of a block in the existing loop structure.  It will not only  *}
463                          {**  allow changes to existing coefficients or factors of the user *}
464                          {**  polynomial, but changes to the order of the polynomial as well*}
465                          {*****************************************************************}
466
467                           Procedure Change_Block;
468
469                          var
470                            i,j,code,Block_num       : integer;
471                            inputerr                 : boolean;
472
473  B-----------------------begin
474  |                        if NBlocks > 0 then  {if there are no blocks to change don't}
475  |  B--------------------begin
476  |  |                      Change:= true;  {sets Change boolean to true}
477  |  |                      Clrscr;
478  |  |                      TextColor(White);
479  |  |                      Center('***Transfer Function Change Routine***',1,2,80);
480  |  |                      HighVideo;
481  |  |                      writeln; writeln(s); writeln;
482  |  |                                                              {prompts for block to change}
483  |  |                      msg('Which Block do you wish to change?',10,6);
484  |  |  B-----------------repeat
485  |  |  |                   Input('N','1',43,6,2,true,F1,F10); {inputs and validates block within range}
486  |  |  |                   val(Answer,Block_num,Code);
487  |  |  |                   if not(Block_num in [1..NBlocks]) then beep(350,150);
488  |  |  E-----------------until Block_num in [1..NBlocks];
489  |  |
490  |  |                      {Initialize the Input_handler holding variables to those of the block to}
491  |  |                      {be changed}
492  |  |
493  |  |                      with Block[Block_num] do     {restore all Filvar[] values prior to display}
494  |  |  B------------------------begin
495  |  |  |
496  |  |  |                        if Feedback then Filvar[01]:='B'
497  |  |  |                                    else Filvar[01]:='F';
498  |  |  |
499  |  |  |                        str(NZeros:1,Filvar[02]);
500  |  |  |                        str(NPoles:1,Filvar[03]);
501  |  |  |                        K := K *(LeadDenCoeff/LeadNumCoeff);  {de-normalize poly for display}
502  |  |  |                        str(K:10:2,  Filvar[04]);
503  |  |  |
504  |  |  |                        if Factored then Filvar[05]:= 'F'
505  |  |  |                                    else Filvar[05]:= 'C';
506  |  |  |
507  |  |  E------------------------end;
508  |  |  B---------------------repeat
509  |  |  |                      Block_Input(Block[Block_Num],Block_Num,inputerr); {Call Block_Input to change}
510  |  |  E---------------------until not(inputerr);
511  |  |                      if Block[Block_num].Feedback then
512  |  |  B-------------------------begin
```

```
513 | | |                          msg('Is the Loop FeedBack Negative (N) or Positive (P)?',1,10);
514 | | |
515 | | | B-----------------------repeat     {validate input answer}
516 | | | |                         if NegFeedBack then ch:= 'N';
517 | | | |                         Input('A',ch,55,10,2,true,F1,F10);
518 | | | |                         ch:=copy(answer,1,1);
519 | | | |                         if not (ch in ['N','P']) then beep(350,150);
520 | | | E-----------------------until ch in ['N','P'];
521 | | |
522 | | |                          if ch ='N' then NegFeedback := true
523 | | |                                    else NegFeedback := false;
524 | | E--------------------------end;
525 | |
526 | |                            Make_Geq; {recalculate equivalent block after changes are made}
527 | E------------------------end;
528 E--------------------------end;
529
530
531
532
533
534                             {*****************************************************************}
535                             {**  The following procedure adds a block to the existing loop by  **}
536                             {**  incrementing the block counter, prompting for all required    **}
537                             {**  information about the block, and then forming the new Geq.    **}
538                             {*****************************************************************}
539
540                             Procedure Add_Block;
541                             var
542                               Inputerr           : boolean;
543
544 B--------------------------Begin
545 |                             Change := false;
546 |                             NBlocks := NBlocks + 1; {pick next block}
547 | B------------------------repeat
548 | |                           Block_Input(Block[NBlocks],NBlocks,Inputerr);{input next block}
549 | E------------------------until not(inputerr);
550 |                             if Block[NBlocks].Feedback then
551 | B----------------------------begin
552 | |                             msg('Is the Loop FeedBack Negative (N) or Positive (P)?',1,10);
553 | |
554 | | B-------------------------repeat     {validate input answer}
555 | | |                           Input('A','',55,10,2,true,F1,F10);
556 | | |                           ch:=copy(answer,1,1);
557 | | |                           if not (ch in ['N','P']) then beep(350,150);
558 | | E-------------------------until ch in ['N','P'];
559 | |
560 | |                            if ch ='N' then NegFeedback := true
561 | |                                      else NegFeedback := false;
562 | E----------------------------end;
563 |
564 |                              Make_Geq; {compute new equivalent block}
565 E------------------------end;
566
567
568
569                             {*****************************************************************}
```

```
570                          {** The following procedure deletes a block from the current loop  **)
571                          {** and shuffles the block indices to close the "gap" left by the  **)
572                          {** removed block.  Then the new Geq is found.                      **)
573                          {*********************************************************************)
574
575                           Procedure Delete_Block;
576                          var
577                            code,BlockNumber,i     :integer;
578
579 B------------------------Begin
580 |                         if NBlocks <> 0 then
581 |  B----------------------begin
582 |  |                       clrscr; Highvideo;                        (prompt for block to delete)
583 |  |                       msg('Which Block do you wish to delete from this loop? [" 0 " to cancel]',1,10);
584 |  |                       input('N','',75,10,1,true,F1,F10);
585 |  |                       val(answer,BlockNumber,code);
586 |  |                       if BlockNumber in [1..NBlocks] then    (if legal block)
587 |  |  B-------------------------begin
588 |  |  |                          NBlocks:=NBlocks-1;  (remove block and adjust block indices accordingly)
589 |  |  |                          for i:= BlockNumber to NBLocks do  Block[i]:=Block[i+1];
590 |  |  |                           Make_Geq;  (compute new equivalent block after deletion)
591 |  |  E-------------------------end;
592 |  E----------------------end;
593 E------------------------end;
594
595
596
597                          {*********************************************************************)
598                          {**  The following procedure allows the user to save the current  **)
599                          {**  loop to a disk file.                                         **)
600                          {*********************************************************************)
601
602                            Procedure Save_Block;
603
604                          var
605                            Blockfile : file of Blocks;
606                            filename  : str20;
607                            Fnr       : integer;
608
609 B------------------------begin
610 |                         clrscr; HighVideo;   (let user change drive if necessary)
611 |                           Msg('###Current Data Drive is    . Press <esc> to change it!###',10,11);
612 |  B----------------------------repeat
613 |  |                              input('A',copy(drive,1,1),36,11,2,true,F1,F10);
614 |  |                              ch:= copy(answer,1,1);
615 |  |                              if not(ch in ['A','B','C','D'])  then beep(350,150);
616 |  E----------------------------until ch in ['A','B','C','D'];
617 |                               Drive := concat(ch,':');
618 |                            clrscr;                              (prompt for filename to store blocks)
619 |                            msg('Diskfile name to store the current blocks?',1,10);
620 |                            writeln; writeln;
621 |                            writeln('###Ensure that your DATA disk is in drive ',Drive,'###');
622 |                            input('A','',45,10,8,true,F1,F10);
623 |                            filename:= concat(Drive,copy(answer,1,8),'.BLX');
624 |
625 |                            Assign(Blockfile,filename); {Open file and save G_eq block first, then loop)
626 |                            REWRITE(Blockfile);         (blocks)
```

```
627  |
628  |                          Write(Blockfile,G_eq);  {store Geq at pointer position 0 in the file}
629  |                                                  {This allows block input routine to load}
630  |                                                  {the equivalent block as a single block}
631  |
632  |                          for pnr:=1 to NBlocks do   {store other block information sequentially}
633  |  B-----------------------begin
634  |  |                           seek(Blockfile,pnr);
635  |  |                           write(Blockfile,Block[pnr]);
636  |  E-----------------------end;
637  |
638  |                          TextColor(Green);  {alert user that loop has been saved}
639  |                          Center('The current problem loop has been stored!',1,20,80);
640  |                          beep(350,150);HighVideo;
641  |
642  |                          delay(200);
643  |
644  |                          close(Blockfile);  {close the file}
645  E--------------------end;
646
647
648
649
650                        {*********************************************************************}
651                        {**  The following procedure allows the user to get the current     **}
652                        {**  loop to a disk file.                                           **}
653                        {*********************************************************************}
654
655                        Procedure Retrieve_Problem;
656                        var
657                          Readfile     : file of Blocks;
658                          filename     : str20;
659                          readerror    : boolean;
660
661  B-----------------------begin
662  |                            Clrscr; HighVideo;  {allow change of disk drive if desired}
663  |                            Msg('***Current Data Drive is    . Press <esc> to change it!***',10,11);
664  |  B--------------------------repeat
665  |  |                           input('A',copy(drive,1,1),36,11,2,true,F1,F10);
666  |  |                           ch:= copy(answer,1,1);
667  |  |                           if not(ch in ['A','B','C','D'])  then beep(350,150);
668  |  E--------------------------until ch in ['A','B','C','D'];
669  |                            Drive := concat(ch,':');
670  |
671  |                                                  {call Directory to display eligible files}
672  |                            Directory(drive,extension,filename,readerror);
673  |
674  |                            if not(readerror) then
675  |  B---------------------------begin
676  |  |                           Assign(Readfile,filename);  {open the file and retrieve contents}
677  |  |                           Reset(Readfile);
678  |  |                           Read(readfile,G_eq);
679  |  |                           NBlocks := 0;
680  |  |                           while not EOF(readfile) do
681  |  |  B------------------------begin
682  |  |  |                           NBlocks:= NBlocks+1;
683  |  |  |                           Seek(readfile,NBlocks);
```

```
684 ¦ ¦ ¦                                Read(readfile,Block[NBlocks]);
685 ¦ ¦ E-------------------------------end;
686 ¦ E-----------------------------end
687 ¦                              else
688 ¦ B-----------------------------begin
689 ¦ ¦                              delay(1500);{else wait for Directory error message and continue}
690 ¦ ¦                              window(1,1,80,25);
691 ¦ ¦                              clrscr;
692 ¦ E-----------------------------end;
693 E-------------------------end;
694
695
696
697
698                          {*****************************************************************}
699                          {** The following procedure provides the secondary program menu   **}
700                          {** for all the block input and manipulation routines.  It is     **}
701                          {** called from the main menu block input/change choice.          **}
702                          {*****************************************************************}
703                          Procedure InputMenu;
704
705                           var
706                            i,Tab     : integer;
707                            Okchoices : set of char;
708
709
710                           procedure MenuItem(pick:char;description:str80;color:integer);
711 B-------------------------begin              {displays menu items in colors desired}
712 ¦                           textcolor(color);
713 ¦                           write('':Tab,'('); textcolor(White); write(pick);
714 ¦                           textcolor(color); writeln(') ',description);
715 E-------------------------end;
716
717
718
719 B-------------------------begin {PROCEDURE INPUTMENU}
720 ¦                             ClrScr; TextColor(White); Finished := false;
721 ¦                             Center('*** INPUT/CHANGE MENU ***  ',1,4,80);
722 ¦                             for i:= 1 to 4 do  writeln('');
723 ¦                             Tab:= 25;
724 ¦                             MenuItem('I','Input Block Transfer Function(s)',green);
725 ¦                             MenuItem('C','Change Block in Current Loop',green);
726 ¦                             writeln;
727 ¦                             MenuItem('A','Add a Block to Current Loop',green);
728 ¦                             MenuItem('D','Delete a Block from Current Loop',green);
729 ¦                             writeln;
730 ¦                             MenuItem('S','Save Current Loop to Disk File',green);
731 ¦                             MenuItem('R','Retrieve Problem from Disk File',green);
732 ¦                             writeln;
733 ¦                             MenuItem('H','Help',blue);
734 ¦                             MenuItem('Q','Exit to Main Menu',lightmagenta);
735 ¦                             TextColor(Green);
736 ¦                             Box(20,2,65,22,6);writeln('');
737 ¦                             Set_Cap_Num('C',' ',' ');Say_Cap_Num;
738 ¦                            TextColor(White); Center('Press Your Selection',21,21,38); LowVideo;
739 ¦                            if NBlocks <= 0 then Okchoices := ['I','A','R','H','Q']
740 ¦                                            else Okchoices := ['I','C','A','D','S','R','H','Q'];
```

```
741 ¦
742 ¦ B-------------------------repeat    {wait for user to input a keypress, check for validity, branch)
743 ¦ ¦                          Option; if not (Ch in OKchoices) then
744 ¦ ¦ B-------------------------------------begin
745 ¦ ¦ ¦                                      Beep(350,150); TextColor(White);
746 ¦ ¦ ¦                                      if NBlocks <= 0 then
747 ¦ ¦ ¦ B-------------------------------------begin
748 ¦ ¦ ¦ ¦                                        msg('WARNING : First INPUT the block description!',1,25);
749 ¦ ¦ ¦ E-------------------------------------end;
750 ¦ ¦ E-------------------------------------end;
751 ¦ ¦
752 ¦ E-------------------------until Ch in OKchoices;
753 ¦ B-------------------------case Ch of
754 ¦ ¦                            '1' : Trans_function_Input;
755 ¦ ¦                            'C' : Change_Block;
756 ¦ ¦                            'A' : Add_Block;
757 ¦ ¦                            'D' : Delete_Block;
758 ¦ ¦                            'S' : Save_Block;
759 ¦ ¦                            'R' : Retrieve_Problem;
760 ¦ ¦                            'H' : InputMenuHelp;
761 ¦ ¦                            'Q' : Finished:= true;
762 ¦ ¦
763 ¦ E-------------------------end;
764 E-------------------------end;
765
766 B-------------------------Begin (Main Program Input)  (due to large size of input routine, compiled as)
767 ¦                         Drive:= 'A';                (a CHAIN program called from CAD.COM.  When     )
768 ¦ B-----------------------Repeat                      (routine is complete, CAD.COM is re-Executed.   )
769 ¦ ¦                       InputMenu;
770 ¦ E-----------------------Until finished;
771 ¦                         Assign(CadFile,'Cad.com');
772 ¦                         Execute(CadFile);
773 E-------------------------end.
```

```
 1                       Program Utility_Menu;  {displays utility menu on screen and branches to
 2                                               user-selected program}
 3
 4                       {$I typedef.sys}
 5                       {$I ut-mod01.inc}
 6                       {$I ut-mod02.inc}
 7                       {$I ut-mod03.inc}
 8                       {$I roots.inc}
 9                       {$I expand.inc}
10                       {$I ShowPoly.inc}
11                       {$I ShowFact.inc}
12                       {$I UserPoly.inc}
13                       {$I UtilHelp.inc}
14
15                       var
16                         through: boolean;
17
18
19
20                       procedure ShowMenu;
21                        var
22                          I,Tab              : integer;
23                          OkChoices          : set of char;
24                          Input_routine      : file;
25
26
27
28                        procedure MenuItem(pick:char;description:str80;color:integer);
29                              {makes writing colors easy}
30  B---------------------------begin
31  |                             textcolor(color);
32  |                             write(''':Tab,'<'); textcolor(White); write(pick);
33  |                             textcolor(color); writeln('> ',description);
34  E---------------------------end;
35
36
37  B---------------------------begin (ShowMenu)
38  |                              ClrScr; TextColor(White);  Through := false;
39  |                              Center('*** UTILITIES MENU ***  ',1,4,80);
40  |                              for I:= 1 to 4 do  writeln('');
41  |                              Tab:= 21;
42  |
43  |                              MenuItem('F','Show Current Loop Blocks (Factored)',green);
44  |                              MenuItem('P','Show Current Loop Blocks (Polynomial)',green);
45  |                              writeln;
46  |                              MenuItem('U','Factor a User-Input Polynomial',green);
47  |                              writeln;
48  |                              MenuItem('H','Help',lightblue);
49  |                              writeln;
50  |                              MenuItem('Q','Exit To Main Menu',lightmagenta);
51  |                              TextColor(Yellow);
52  |                              Box(20,2,65,18,6);writeln('');
53  |                              Set_Cap_Num('C',' ',' ');Say_Cap_Num;
54  |                             TextColor(White); Center('Press Your Selection',21,17,38); LowVideo;
55  |                                       {sets legal choices if no blocks have been entered}
56  |                              If NBlocks <= 0 then OKchoices := ['H','Q','U']
```

```
57  ¦                                          else OKchoices := ['F','P','Q','U','H'];
58  ¦  B-------------------------repeat
59  ¦  ¦                            Option;
60  ¦  ¦                            if not (Ch in OKchoices) then
61  ¦  ¦  B---------------------------begin
62  ¦  ¦  ¦                              Beep(350,150); TextColor(White);
63  ¦  ¦  ¦                               if NBlocks <= 0 then
64  ¦  ¦  ¦  B-----------------------------begin
65  ¦  ¦  ¦  ¦                                msg('WARNING : First INPUT the block description!',1,25);
66  ¦  ¦  ¦  E-----------------------------end;
67  ¦  ¦  E---------------------------end;
68  ¦  ¦
69  ¦  E-------------------------until Ch in OKchoices;
70  ¦
71  ¦  B-------------------------case Ch of
72  ¦  ¦                            'F' : Show_Factored_Roots;
73  ¦  ¦                            'P' : ShowPoly;
74  ¦  ¦                            'U' : User_Polynomial_Roots;
75  ¦  ¦                            'H' : UtilitiesHelp;
76  ¦  ¦  B-------------------------'Q' : Begin
77  ¦  ¦  ¦                              through := true;
78  ¦  ¦  ¦                              assign(cadfile,'cad.com');
79  ¦  ¦  ¦                              execute(cadfile);
80  ¦  ¦  E-----------------------------end;
81  ¦  E-------------------------end;
82  E-----------------------end;
83
84                           {*************************************************************}
85                           {*             Program Starts Execution                      *}
86                           {*************************************************************}
87
88  B-----------------------begin
89  ¦
90  ¦  B-------------------------repeat
91  ¦  ¦                          ShowMenu;
92  ¦  E-------------------------until through;
93  E-----------------------end.
```

```
  1                         {*************************************************************************}
  2                         {**  FREQUENCY_RESPONSE is the driver program for the Bode and Nyquist    **}
  3                         {**  plotting routines.  It simply invokes Bode and, when finished, returns**}
  4                         {**  control back to the main menu.                                       **}
  5                         {*************************************************************************}
  6
  7                         Program Frequency_Response;
  8
  9                         {$I typedef.sys}  {graphics routines}
 10                         {$I graphix.sys}
 11                         {$I kernel.sys}
 12                         {$I windows.sys}
 13                         {$I polygon.hgh}
 14                         {$I axis.hgh}
 15
 16                         {$I UT-MOD01.INC }  {I/O routines}
 17                         {$I UT-MOD02.INC }
 18                         {$I UT-MOD03.INC }
 19
 20                         {$I GrapMenu.inc}   {graph options menu}
 21                         {$I PlotBode.inc}   {Bode plotting routine}
 22                         {$I PlotNyqs.inc}   {Nyquist plotting routine}
 23                         {$I Roots.inc}      {RootFinder routine}
 24
 25                         {$I Bode.inc}       {Bode routine}
 26
 27
 28 B-----------------------begin
 29 |                          BODE;   {call the Bode calculation routine}
 30 |
 31 |                          Assign(cadfile,'CAD.COM');     {re-execute the main menu routines}
 32 |                          Execute(cadfile);
 33 E-----------------------end.
```

```
  1                             procedure Bode;
  2
  3                             var
  4                                 Code,I,Count,NumberDecades,StartDecade,EndDecade : integer;
  5                                 Wf,Wo,Wi,DeltaW                                  : Real;
  6                                 BReal,BImag,AReal,AImag                          : Array[1..20] of real;
  7                                 PlotArray1, PlotArray2 , MagPhaseArray           : PlotArray;
  8                                 ZMagn,ZPhase,PMagn,PPhase,Phase                  : real;
  9                                 TempX,TempY                                      : real;
 10                                 CLOSEQ                                           : blocks;
 11                                 temp                                             : char;
 12                                 OpenLoop, Nyquist ,BigPic                        : boolean;
 13
 14
 15                             function Log(X:real):real;    {computes the base-10 logarithm of X}
 16  B--------------------------Begin
 17  |                              If X=0 then Log:=0 else
 18  |                              Log := Ln(X)/Ln(10);
 19  E--------------------------End;
 20
 21                             function Expon(Y,X:real):real;  {computes Y raised to X power}
 22  B--------------------------Begin
 23  |                               Expon := exp( X * (ln(Y)));
 24  E--------------------------end;
 25
 26
 27
 28  B------------------------begin
 29  |                              ClrScr; TextColor(White);
 30  |                              Center('***Bode/Nyquist Plotting Routine***',1,2,80);
 31  |                              writeln;writeln;
 32  |                              BigPic:= false;
 33  |                              TextColor(Green);
 34  |
 35  |                              Msg('Bode (B) or Nyquist (N) Plot?',5,5);
 36  |
 37  |  B-------------------------repeat
 38  |  |                             input('A','',45,5,2,true,F1,F10);            {sets flag Nyquist  if}
 39  |  |                             temp := copy(answer,1,1);                    {user selects the Nyquist}
 40  |  |                             if not(temp in ['B','N']) then beep(350,150); {option for plot }
 41  |  E-------------------------until temp in ['B','N'];
 42  |                              if temp = 'N' then Nyquist  := true
 43  |                                            else Nyquist  := false;
 44  |
 45  |                              if Nyquist then
 46  |  B---------------------------begin
 47  |  |                              Msg('See the BIG (B) picture, or select your own Window (W) size?',5,6);
 48  |  |
 49  |  |  B-----------------------repeat
 50  |  |  |                           input('A','',65,6,2,true,F1,F10);
 51  |  |  |                           temp := copy(answer,1,1);
 52  |  |  |                           if not(temp in ['B','W']) then beep(350,150);
 53  |  |  E-----------------------until temp in ['B','W'];
 54  |  |                             if temp = 'B' then BigPic   := true
 55  |  |                                           else BigPic   := false;
 56  |  E---------------------------end;
```

```
57  |
58  |                              Msg('Open (O) or Closed (C) Loop Plot?',5,7);
59  |
60  |  B--------------------------repeat
61  |  |                             input('A','',45,7,2,true,F1,F10);            {sets flag OpenLoop if}
62  |  |                             temp := copy(answer,1,1);                    {user selects the open}
63  |  |                             if not(temp in ['O','C']) then beep(350,150);  {loop option for plot }
64  |  E--------------------------until temp in ['O','C'];
65  |                              if temp = 'O' then OpenLoop := true
66  |                                            else OpenLoop := false;
67  |
68  |                            if not(BigPic) then
69  |  B--------------------------begin
70  |  |                           Msg('What is the first frequency to be plotted?',5,9);
71  |  |                           Msg('(e.g. .01, .001, 1000, etc.)',10,10);
72  |  |                           Input('N','',45,10,8,true,F1,F10);
73  |  |                           Val(answer,Wo,code);               {Wo is the first plotted freq}
74  |  |
75  |  |                           Msg('How many decades do you want plotted?',5,13);
76  |  |                           Input('N','',45,13,2,true,F1,F10);
77  |  |                           Val(answer,NumberDecades,code);
78  |  E--------------------------end
79  |                            else
80  |  B--------------------------begin
81  |  |                           Wo:=0.001;
82  |  |                           NumberDecades:=8;
83  |  E--------------------------end;
84  |
85  |                            ClrScr;TextColor(White);
86  |                            Center('*** Computing Points for Bode/Nyquist Plot - Please Wait ***',1,12,80);
87  |                            TextColor(Yellow);
88  |
89  |                            with G_eq do          {compute closed-loop (unity feedback) G equivalent)
90  |  B--------------------------begin                {called CLGEQ -- which is also a blocks type record}
91  |  |
92  |  |                              CLGEQ.K := K;          {closed loop K same as open loop K)
93  |  |                              CLGEQ.NZEROS:=NZeros;  {as are the zeros of the function )
94  |  |
95  |  |                              for i:=1 to maxorder do CLGEQ.DenCoeff[i] := 0.0;   {initialize)
96  |  |
97  |  |                              for i:= 1 to NZeros do
98  |  |  B---------------------------begin
99  |  |  |                            CLGEQ.RealPartZero[i] := RealPartZero[i];   {same zeros)
100 |  |  |                            CLGEQ.ImagPartZero[i] := ImagPartZero[i];
101 |  |  E---------------------------end;
102 |  |
103 |  |                              for i:=1 to NZeros + 1 do
104 |  |  B---------------------------begin
105 |  |  |                            CLGEQ.DenCoeff[i] := NumCoeff[i] * K;   {C.L. denominator equals the )
106 |  |  |                            CLGEQ.NumCoeff[i] := NumCoeff[i];       {sum of open loop denominator)
107 |  |  E---------------------------end;                                     {and K times O.L. numerator  )
108 |  |
109 |  |                              for i:=1 to NPoles + 1 do CLGEQ.DenCoeff[i] := CLGEQ.DenCoeff[i] +
110 |  |                                                                             DenCoeff[i];
111 |  |
112 |  |
113 |  |                              if NPoles > NZeros then CLGEQ.NPOLES:=NPoles  {NPoles should always be)
```

```
114 ¦ ¦                                          else CLGEQ.NPOLES:=NZeros; {greater, but to be safe}
115 ¦ ¦
116 ¦ ¦                                 {compute new denominator roots}
117 ¦ ¦                              rootfinder(CLGEQ.NPOLES,CLGEQ.DenCoeff,CLGEQ.RealPartPole,
118 ¦ ¦                                                          CLGEQ.ImagPartPole,0,0);
119 ¦ E-------------------------end; {with}
120 ¦
121 ¦                             StartDecade := trunc(Log(Wo));          {compute linear scale to plot }
122 ¦                             EndDecade := StartDecade + NumberDecades;  {log numbers. Also figure step}
123 ¦                             Wf := Wo * Expon(10.0,NumberDecades);
124 ¦                             DeltaW := Expon((Wf/Wo),0.0125);
125 ¦                             Wi := Wo;
126 ¦
127 ¦
128 ¦                             for Count := 1 to 81 do                 {do 100 iterations...arbitrary #}
129 ¦ B-------------------------Begin
130 ¦ ¦
131 ¦ ¦                           if OpenLoop then          {compute bode numbers with G_eq if openloop}
132 ¦ ¦                            with G_eq do             {and later with CLGEQ if closed loop       }
133 ¦ ¦ B-----------------------begin
134 ¦ ¦ ¦
135 ¦ ¦ ¦                           ZMagn:=1.0;ZPhase:=0.0;PMagn:=1.0;PPhase:=0.0;  {initialize}
136 ¦ ¦ ¦
137 ¦ ¦ ¦                           for I := 1 to NZeros do   {compute magn and phase of zeros for freq step}
138 ¦ ¦ ¦ B-------------------------begin
139 ¦ ¦ ¦ ¦                             ZMagn:=ZMagn * Sqrt(Sqr(RealPartZero[I])+Sqr(Wi-ImagPartZero[I]));
140 ¦ ¦ ¦ ¦                             if RealPartZero[I] = 0.0 then ZPhase:=ZPhase+pi/2.0 else
141 ¦ ¦ ¦ ¦                             ZPhase:=ZPhase+ArcTan((Wi-ImagPartZero[I])/(-RealPartZero[I]));
142 ¦ ¦ ¦ E-------------------------end;
143 ¦ ¦ ¦
144 ¦ ¦ ¦                           for I := 1 to NPoles do   {compute magn and phase of poles for freq step}
145 ¦ ¦ ¦ B-------------------------begin
146 ¦ ¦ ¦ ¦                             PMagn:=PMagn * Sqrt(Sqr(RealPartPole[I])+Sqr(Wi-ImagPartPole[I]));
147 ¦ ¦ ¦ ¦                             if RealPartPole[I] = 0.0 then PPhase:=PPhase+pi/2.0 else
148 ¦ ¦ ¦ ¦                             PPhase:=PPhase+ArcTan((Wi-ImagPartPole[I])/(-RealPartPole[I]));
149 ¦ ¦ ¦ E-------------------------end;
150 ¦ ¦ ¦
151 ¦ ¦ ¦ B---------------------If Nyquist then Begin
152 ¦ ¦ ¦ ¦                                   Phase := Frac((ZPhase - PPhase)/(2*pi)) * (2*pi);
153 ¦ ¦ ¦ ¦                                            {Phase "modulo" 2Pi}
154 ¦ ¦ ¦ ¦                                   TempX := abs((K*ZMagn/PMagn)*cos(Phase));
155 ¦ ¦ ¦ ¦                                   TempY := abs((K*ZMagn/PMagn)*sin(Phase));
156 ¦ ¦ ¦ ¦                                  if (BigPic) and (TempX > 100) then TempX := 100;
157 ¦ ¦ ¦ ¦                                  if (BigPic) and (TempY > 100) then TempY := 100;
158 ¦ ¦ ¦ ¦
159 ¦ ¦ ¦ ¦                                   If Phase<0 then Phase:= Phase+(2*pi);
160 ¦ ¦ ¦ ¦                                   If ((Phase)>(pi/2)) and (Phase<(3*pi/2))) then
161 ¦ ¦ ¦ ¦                                                                    TempX:= -TempX;
162 ¦ ¦ ¦ ¦                                   If ((Phase>pi) and (Phase<(2*pi))) then TempY := -TempY;
163 ¦ ¦ ¦ ¦                                   MagPhaseArray[count,2] := Phase;
164 ¦ ¦ ¦ ¦                                   MagPhaseArray[count,1]  := K*ZMagn/Pmagn;
165 ¦ ¦ ¦ ¦                                   PlotArray1[Count,1] := TempX;
166 ¦ ¦ ¦ ¦                                   PlotArray1[Count,2] := -TempY;
167 ¦ ¦ ¦ E-----------------------------------end
168 ¦ ¦ ¦                                 else
169 ¦ ¦ ¦ B-----------------------------------Begin
170 ¦ ¦ ¦ ¦                                   PlotArray1[Count,1] := Log(Wi);      {fill plotting matrix}
```

```
171 ¦ ¦ ¦ ¦                                                              {with magnitude values}
172 ¦ ¦ ¦ ¦                                  PlotArray1[Count,2] := 20*Log(K*(ZMagn/FMagn));
173 ¦ ¦ ¦ ¦
174 ¦ ¦ ¦ ¦                                  PlotArray2[Count,1] := Log(Wi);       {fill phase matrix}
175 ¦ ¦ ¦ ¦                                  PlotArray2[Count,2] := (180/pi)*(ZPhase-FPhase);
176 ¦ ¦ ¦ ¦                                                  {next stmt covers freq wrap-around}
177 ¦ ¦ ¦ ¦                                  if PlotArray2[Count,2] > 0 then
178 ¦ ¦ ¦ ¦                                            PlotArray2[count,2]:=PlotArray2[count,2]-360;
179 ¦ ¦ ¦ E-------------------------------------end;
180 ¦ ¦ ¦                          Wi := Wi + DeltaW;        {increment freq step}
181 ¦ ¦ E------------------------end
182 ¦ ¦
183 ¦ ¦                        else with CLOEQ do     {perform same steps as above if closed loop requested}
184 ¦ ¦ B------------------------begin
185 ¦ ¦ ¦                          ZMagn:=1.0;ZPhase:=0.0;FMagn:=1.0;FPhase:=0.0;
186 ¦ ¦ ¦                          for I := 1 to NZeros do
187 ¦ ¦ ¦ B--------------------------begin
188 ¦ ¦ ¦ ¦                            ZMagn:=ZMagn * Sqrt(Sqr(RealPartZero[I])+Sqr(Wi-ImagPartZero[I]));
189 ¦ ¦ ¦ ¦                            if RealPartZero[I] = 0.0 then ZPhase:=ZPhase+pi/2.0 else
190 ¦ ¦ ¦ ¦                            ZPhase:=ZPhase+ArcTan((Wi-ImagPartZero[I])/(-RealPartZero[I]));
191 ¦ ¦ ¦ E----------------------end;
192 ¦ ¦ ¦
193 ¦ ¦ ¦                          for I := 1 to NPoles do
194 ¦ ¦ ¦ B--------------------------begin
195 ¦ ¦ ¦ ¦                            PMagn:=PMagn * Sqrt(Sqr(RealPartPole[I])+Sqr(Wi-ImagPartPole[I]));
196 ¦ ¦ ¦ ¦                            if RealPartPole[I] = 0.0 then PPhase:=PPhase+pi/2.0 else
197 ¦ ¦ ¦ ¦                            PPhase:=PPhase+ArcTan((Wi-ImagPartPole[I])/(-RealPartPole[I]));
198 ¦ ¦ ¦ E----------------------end;
199 ¦ ¦ ¦
200 ¦ ¦ ¦ B----------------------If Nyquist then Begin
201 ¦ ¦ ¦ ¦                                  Phase := Frac((ZPhase - PPhase)/(2*pi)) * (2*pi);
202 ¦ ¦ ¦ ¦                                          {Phase "modulo" 2Pi}
203 ¦ ¦ ¦ ¦                                  TempX := abs((K*ZMagn/PMagn)*cos(Phase));
204 ¦ ¦ ¦ ¦                                  TempY := abs((K*ZMagn/PMagn)*sin(Phase));
205 ¦ ¦ ¦ ¦                                  if (BigPic) and (TempX > 100) then TempX := 100;
206 ¦ ¦ ¦ ¦                                  if (BigPic) and (TempY > 100) then TempY := 100;
207 ¦ ¦ ¦ ¦
208 ¦ ¦ ¦ ¦                                  If Phase<0 then Phase:= Phase+(2*pi);
209 ¦ ¦ ¦ ¦                                  If ((Phase>(pi/2)) and (Phase<(3*pi/2))) then
210 ¦ ¦ ¦ ¦                                                                      TempX:= -TempX;
211 ¦ ¦ ¦ ¦                                  If ((Phase>pi) and (Phase<2*pi)) then TempY := -TempY;
212 ¦ ¦ ¦ ¦
213 ¦ ¦ ¦ ¦                                  PlotArray1[Count,1] := TempX;
214 ¦ ¦ ¦ ¦                                  PlotArray1[Count,2] := -TempY;
215 ¦ ¦ ¦ E-------------------------------------end
216 ¦ ¦ ¦                                 else
217 ¦ ¦ ¦ B-------------------------------------Begin
218 ¦ ¦ ¦ ¦                                  PlotArray1[Count,1] := Log(Wi);      {fill plotting matrix}
219 ¦ ¦ ¦ ¦                                                                    {with magnitude values}
220 ¦ ¦ ¦ ¦                                  PlotArray1[Count,2] := 20*Log(K*(ZMagn/PMagn));
221 ¦ ¦ ¦ ¦
222 ¦ ¦ ¦ ¦                                  PlotArray2[Count,1] := Log(Wi);       {fill phase matrix}
223 ¦ ¦ ¦ ¦                                  PlotArray2[Count,2] := (180/pi)*(ZPhase-FPhase);
224 ¦ ¦ ¦ ¦                                                  {next stmt covers freq wrap-around}
225 ¦ ¦ ¦ ¦                                  if PlotArray2[Count,2] > 0 then
226 ¦ ¦ ¦ ¦                                            PlotArray2[count,2]:=PlotArray2[count,2]-360;
227 ¦ ¦ ¦ E-------------------------------------end;
```

```
228  ¦  ¦  ¦                    Wi := Wi + DeltaW;
229  ¦  ¦  E--------------------end;
230  ¦  E-----------------------end;
231  ¦                         if Nyquist then
232  ¦                              Plot_Nyquist(StartDecade,EndDecade,NumberDecades,PlotArray1,
233  ¦                                           MagPhaseArray,BigPic,OpenLoop)
234  ¦
235  ¦
236  ¦                                 else
237  ¦                              PlotBode(StartDecade,EndDecade,NumberDecades,PlotArray1,PlotArray2,
238  ¦                                                    OpenLoop );
239  ¦
240  E-------------------------end;
```

```
 1                          procedure PlotBode(StartDecade,EndDecade,NumberDecades:Integer;
 2                                             PlotArray1,PlotArray2:PlotArray;OpenLoop: Boolean);
 3
 4                          {*****************************************************************}
 5                          {**  Procedure to Plot the LIN-LOG chart for the Bode diagram.  **}
 6                          {*****************************************************************}
 7
 8
 9                          const
10                              MagnArray: array[1..12] of char = ('M','A','G','N','I','T','U','D','E',' ','d','B');
11                              PhasArray: array[1..12] of char = ('P','H','A','S','E',' ','d','e','g',' ',' ',' ');
12                              FreqArray: string[19] = 'FREQUENCY (rad/sec)';
13
14                          var i,j,n       :integer;
15                              ch          :char;
16                              x1,x2       :integer;
17                              Delta       :real;
18                              MagLabel    : string[3];
19                              PhsLabel    : string[4];
20                              DecLabel    : string[3];
21                              Title1,
22                              Title2      : string[80];
23                              DumpGraph   : Boolean;
24                              w           : real;
25                              quit        : Boolean;
26                              list        : text;
27
28                          function Log(X:real):real;
29  B-----------------------------Begin
30  |                           If X=0 then Log:=0 else
31  |                           Log := Ln(X)/Ln(10);
32  E-----------------------------End;
33
34                          function Expon(Y,X:real):real;  {computes Y raised to X power}
35  B-----------------------------Begin
36  |                             Expon := exp( X * (ln(Y)));
37  E-----------------------------end;
38
39
40                          Procedure PrintGraphData;   {prints numbers to a file or printer}
41  B-----------------------------Begin
42  |                           LeaveGraphic;
43  |                           Clrscr;
44  |                           Center('*** TURN ON PRINTER AND ALIGN PAPER ***',1,10,80);
45  |                           TextColor(green);
46  |                           msg('press <P> to continue, <F> to list to file*, <Q> to quit print',1,13);
47  |                           TextColor(white);
48  |                           msg('*File option prints numbers to a file named "BODE.NUM"',1,17);
49  |                           msg('on the current drive.  Browse this file off-line using the DOS "type" command.',1,18);
50  |
51  |  B--------------------------repeat
52  |  |                          Read(kbd,ch);
53  |  |                            If (ch = 'C') or (ch = 'c') or (ch = 'P') or (ch = 'p') then
54  |  |  B-------------------------begin
55  |  |  |                           If (ch = 'F') or (ch = 'f') then
56  |  |  |  B----------------------------begin
```

```
 57 | | | |                        assign(list,'Bode.NUM');   {assign file}
 58 | | | |                        rewrite(list);
 59 | | | E------------------------end
 60 | | |                       else
 61 | | | B------------------------begin
 62 | | | |                        assign(list,'LST:');       {otherwise assign printer}
 63 | | | |                        rewrite(list);
 64 | | | E------------------------end;
 65 | | |
 66 | | |                        Title1:=('      w (rad)             Gain (db)              Phase (deg)');
 67 | | |                        Title2:=('-------------------------------------------------------------------------');
 68 | | |                        writeln(list,Title1);
 69 | | |                        writeln(list,Title2);
 70 | | |                        writeln(list); writeln(list);
 71 | | |                        for i:= 1 to 81 do
 72 | | | B----------------------begin
 73 | | | |                         w := expon(10.0,PlotArray1[i,1]);
 74 | | | |                         writeln(list,'   ',w:11:3,'             ',PlotArray1[i,2]:8:3,'                 ',PlotArray2[i,
:7:3);
 75 | | | |                        if i= 50 then
 76 | | | | B----------------------begin
 77 | | | | |                         write(list,chr(12));
 78 | | | | |                         writeln(list);
 79 | | | | |                         writeln(list,Title1);
 80 | | | | |                         writeln(list,Title2); writeln(list); writeln(list);
 81 | | | | E----------------------end;
 82 | | | E------------------------end;
 83 | | E--------------------------end;
 84 | E----------------------------until ch in ['F','f','Q','q','P','p'];
 85 |                              EnterGraphic;   {when finished printing, go back to graphics mode and}
 86 |                              swapscreen;     {display graph}
 87 |                              close(list);
 88 E----------------------------end;
 89
 90
 91
 92
 93
 94 B--------------------------begin
 95 |                            initgraphic;                          {set-up windows for display}
 96 |                            DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
 97 |                            DefineWindow(2,5,15,XMaxGlb-5,YMaxGlb-15);
 98 |                            DefineWindow(3,5,15,XmaxGlb-5,YMaxGlb-15);
 99 |                            DefineWorld(1,0,0,100,100);
100 |                            DefineWorld(2,StartDecade,60,EndDecade,-60);
101 |                            DefineWorld(3,StartDecade,0,EndDecade,-360);
102 |                            SelectWorld(1);
103 |                            SelectWindow(1);
104 |                            SetBackground(0);
105 |                            SelectWorld(2);
106 |                            SelectWindow(2);
107 |                            DrawBorder;
108 |
109 |                            SetLineStyle(1);
110 |                            For I:=1 to 5 do      {draw horizontal graph lines}
111 |                               DrawLine(StartDecade,-60+(20*I),EndDecade,-60+(20*I));
112 |
113 |                            For J:=0 to NumberDecades-1 do    {draw vertical logarithmic graph lines}
```

```
114 ¦ B----------------------------Begin
115 ¦ ¦                            For I:= 1 to 10 do
116 ¦ ¦ B--------------------------Begin
117 ¦ ¦ ¦                              Delta:=StartDecade + (Log(I) + J);
118 ¦ ¦ ¦                              Drawline(Delta,-60,Delta,60);
119 ¦ ¦ E--------------------------end;
120 ¦ E------------------------end;
121 ¦
122 ¦                          SelectWindow(1);        {y-axis titles}
123 ¦                          For I:= 1 to 12 do
124 ¦ B--------------------------Begin
125 ¦ ¦                            DrawText(5,55+6*I,1,MagnArray[I]);
126 ¦ ¦                            DrawText(630,60+6*I,1,PhasArray[I]);
127 ¦ E------------------------end;
128 ¦
129 ¦                          DrawText(250,195,1,FreqArray);         {x-axis title}
130 ¦
131 ¦                          For I:= 0 to 6 do                    {y-axis scale label}
132 ¦ B--------------------------Begin
133 ¦ ¦                            Str(60-20*I:3,MagLabel);
134 ¦ ¦                            DrawText(12,13+28*I,1, MagLabel);
135 ¦ ¦                            Str(0-60*I:4,PhsLabel);
136 ¦ ¦                            DrawText(600,13+28*I,1,PhsLabel);
137 ¦ E------------------------end;
138 ¦
139 ¦                          For I:= 0 to NumberDecades do          {label the logarithmic scale}
140 ¦ B--------------------------Begin
141 ¦ ¦                            Str(Trunc(StartDecade)+I:3,DecLabel);
142 ¦ ¦                            DrawText(36+(570 div NumberDecades) *I,186,1,DecLabel);
143 ¦ ¦                            DrawText(30+(570 div NumberDecades) *I,190,1,'10');
144 ¦ E------------------------end;
145 ¦
146 ¦                          SetLineStyle(0);
147 ¦                          SelectWindow(2);
148 ¦                          DrawPolygon(PlotArray1,1,-81,0,1,0);   {plot the magnitude}
149 ¦                          SelectWorld(3);
150 ¦                          SelectWindow(3);
151 ¦                          SetLineStyle(3);
152 ¦                          DrawPolygon(PlotArray2,1,-81,0,1,0);   {plot the phase}
153 ¦                          copyscreen;           {save screen to memory}
154 ¦ E----------------------repeat until keypressed;
155 ¦                          quit := false;
156 ¦ B----------------------repeat                 {call for graph options menu}
157 ¦ ¦                           if OpenLoop then Graph_Menu('Open Loop Bode Plot',DumpGraph,quit)
158 ¦ ¦                                       else Graph_Menu('Closed Loop Bode',DumpGraph,quit);
159 ¦ ¦                           If DumpGraph then PrintGraphData;  {dump numbers if desired}
160 ¦ E----------------------until quit;
161 ¦                          LeaveGraphic;    {leave graphics mode}
162 E------------------------end;
```

```
                          {*************************************************************************}
                          {**  Plot_Nyquist is a routine to draw the Nyquist plot from the data     **}
                          {**  generated in the Bode procedure.                                     **}
                          {*************************************************************************}

                          Procedure Plot_Nyquist(StartDecade, EndDecade, NumberDecades:Integer;
                                                 PlotArray1,MagPhaseArray:PlotArray;
                                                 BigPic,OpenLoop:Boolean);

                          var
                            Xint,Yint,
                            i,j,n,code : integer;
                            Ymin,Ymax,
                            Xmin,Xmax  : real;
                            XLabel,
                            YLabel     : String[3];
                            Title1,
                            Title2     : String[80];
                            DumpGraph,
                            quit       : Boolean;
                            w          : real;
                            XminL,YminL: Real;
                            GraphWidthX,
                            GraphWidthY: Real;
                            Xexponent,
                            Yexponent  : integer;
                            XexpLabel,
                            YexpLabel  : string[3];
                            GraphArray : PlotArray;
                            List       : text;


                          function Expon(Y,X:real):real;  {computes Y raised to X power)
B-------------------------------Begin
|                                 Expon := exp( X * (ln(Y)));
E-------------------------------end;


                          Procedure PrintGraphData;     {dump data used to make graph to printer)
B-------------------------------Begin
|                                 LeaveGraphic;
|                                 Clrscr;
|                                 Center('*** TURN ON PRINTER AND ALIGN PAPER ***',1,10,80);
|                                 TextColor(green);
|                                 msg('press <P> to Print, <F> to list to file *,<Q> to quit print',1,13);
|                                 TextColor(white);
|  B----------------------------repeat
|  |                              msg('*File option prints numbers to a file named "NYQUIST.NUM"',1,17);
|  |                              msg('on the current drive.  Browse this file off-line using DOS "type" command.',1,18);
|  |
|  |
|  |                               Read(kbd,ch);
|  |                                 If (ch = 'F') or (ch = 'f') or (ch = 'p') or (ch = 'P') then
|  |  B-------------------------begin
```

```
57  |  |  |                          if (ch = 'F') or (ch = 'f') then
58  |  |  |  B-----------------------begin
59  |  |  |  |                          assign(list,'Nyquist.NUM');
60  |  |  |  |                          rewrite(list);
61  |  |  |  E-----------------------end
62  |  |  |                          else
63  |  |  |  B-----------------------begin
64  |  |  |  |                          assign(list,'LST:');
65  |  |  |  |                          rewrite(list);
66  |  |  |  E-----------------------end;
67  |  |  |
68  |  |  |                          Title1:=('   w (rad)      Magnitude          Phase (rad)         Xplot         YPlot');
69  |  |  |                          Title2:=('-------------------------------------------------------------------------------');
70  |  |  |                          writeln(list,Title1); Writeln(list,Title2);
71  |  |  |                          writeln(list); writeln(list);
72  |  |  |                          for i:= 1 to 81 do
73  |  |  |  B-----------------------begin
74  |  |  |  |                          w := expon(10.0,PlotArray1[i,1]);
75  |  |  |  |                          writeln(list,w:9:3,'     ',MagPhaseArray[i,1]:11:3,'       ',MagPhaseArray[i,2]:11:3,'
76  |  |  |  |                                                PlotArray1[i,1]:10:3,'    ',-PlotArray1[i,2]:10:3);
77  |  |  |  |                          if i= 50 then
78  |  |  |  |  B--------------------------begin
79  |  |  |  |  |                          writeln(list);
80  |  |  |  |  |                          write(list,chr(12));
81  |  |  |  |  |                          writeln(list,Title1);
82  |  |  |  |  |                          writeln(list,Title2);
83  |  |  |  |  |                          writeln(list); writeln(list);
84  |  |  |  |  E--------------------------end;
85  |  |  |  E-----------------------end;
86  |  |  E--------------------------end;
87  |  E-----------------------until ch in ['F','f','Q','q','P','p'];
88  |                             EnterGraphic;
89  |                             swapscreen;
90  |                             close(list);
91  E-----------------------end;
92
93
94  B-----------------------Begin (Plot_Nyquist)            (prompt for window parameters)
95  |                          if not(BigPic) then
96  |  B-----------------------begin
97  |  |                          P[5]:= '1511N00505-010101';
98  |  |                          P[6]:= '1512N00506-010101';
99  |  |
100 |  |                          Clrscr; TextColor(LightBlue);
101 |  |                          Center('*** NYQUIST PLOTTING ROUTINE ***',1,2,80);
102 |  |                          HighVideo;
103 |  |                          FillChar(s,100,#205);S:= copy(s,1,80);
104 |  |                          writeln;writeln(s);writeln;
105 |  |                          msg('Viewing Coordinates for Nyquist Plot',1,6);
106 |  |                          TextColor(white);
107 |  |                          msg('Enter X and Y values (graph will span  -X to X, -Y to Y)',1,7);
108 |  |                          TextColor(yellow);
109 |  |                          writeln;writeln;writeln;writeln;
110 |  |
111 |  |                          writeln('X-Maximum: ');
112 |  |                          writeln('Y-Maximum: ');
113 |  |
```

```
114 ¦ ¦                          Input_Handler('N0506',Escape);
115 ¦ ¦                          writeln;writeln;
116 ¦ ¦                          writeln('Any changes to these parameters? (Y or N):');
117 ¦ ¦                          Input('A','',45,14,2,true,F1,F10);
118 ¦ ¦                          if (answer='Y') or (F1) then  Input_Handler('C0506',Escape);
119 ¦ ¦                          Val(filvar[5],Xmax,code);
120 ¦ ¦                          Val(filvar[6],Ymax,code);
121 ¦ ¦                          Xmin:=-Xmax; Ymin:=-Ymax;
122 ¦ E--------------------------end
123 ¦                           else
124 ¦ B--------------------------begin
125 ¦ ¦                          Xmin:= -50;    {set default values for "big picture" plot}
126 ¦ ¦                          Xmax:=  50;
127 ¦ ¦                          Ymin:= -50;
128 ¦ ¦                          Ymax:=  50;
129 ¦ E--------------------------end;
130 ¦
131 ¦                            INITGRAPHIC;                    {define world/window and draw X&Y axes}
132 ¦                            DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
133 ¦                            DefineWindow(2,0,0,XMaxGlb,YMaxGlb);
134 ¦                            DefineWorld(1,Xmin,Ymin,Xmax,Ymax);
135 ¦                            DefineWorld(2,0,0,100,100);
136 ¦                            SelectWorld(1);SelectWindow(1);DrawBorder;
137 ¦                            SETCLIPPINGON;
138 ¦                            DrawLine(Xmin,0,Xmax,0);
139 ¦                            DrawLine(0,Ymin,0,Ymax);
140 ¦                            n:=1;
141 ¦                            for i:=1 to 80 do
142 ¦                            if (abs(plotarray1[i,1]) > Xmax) or (abs(plotarray1[i,2]) > Ymax) then
143 ¦                               n:= n+1;
144 ¦                            if n<>1 then n:= n-1;   {use 1 extra point beyond graph border}
145 ¦
146 ¦                            DrawPolygon(PlotArray1,n,-80,0,1,0);  {draw graph on screen}
147 ¦
148 ¦
149 ¦
150 ¦                                                {compute X & Y non-fraction labels with exponent}
151 ¦                            Xexponent := 0;
152 ¦                            graphwidthX := Xmax - Xmin; XminL:= Xmin;
153 ¦                            while graphwidthX < 10 do
154 ¦ B--------------------------begin
155 ¦ ¦                            graphwidthX := graphwidthX * 10.0;
156 ¦ ¦                            XminL := XminL * 10.0;
157 ¦ ¦                            Xexponent := Xexponent -1;
158 ¦ E--------------------------end;
159 ¦
160 ¦                            Yexponent := 0;
161 ¦                            graphwidthY := Ymax - Ymin; YminL := Ymin;
162 ¦                            while graphwidthY < 10 do
163 ¦ B--------------------------begin
164 ¦ ¦                            graphwidthY := graphwidthY * 10.0;
165 ¦ ¦                            YminL:=YminL *10.0;
166 ¦ ¦                            Yexponent := Yexponent -1;
167 ¦ E--------------------------end;
168 ¦
169 ¦                           str(Xexponent:3,XexpLabel);
170 ¦                           str(Yexponent:3,YexpLabel);
```

```
171  ¦
172  ¦                              Xint:=Round((XmaxGlb+1)/10);
173  ¦                              Yint:=Round((YMaxGlb+1)/10);
174  ¦
175  ¦                              SelectWorld(2);
176  ¦                              SelectWindow(2);
177  ¦                              For i:= 1 to 9 do            {label X,Y axes with scale)
178  ¦  B-------------------------------Begin
179  ¦  ¦                                str(Round(XminL+GraphWidthX/10 * i):3,XLabel);
180  ¦  ¦                                str(Round(YminL+GraphWidthY/10 * i):3,YLabel);
181  ¦  ¦                                DrawTextW(10 * i,47,1,XLabel);
182  ¦  ¦                                if i <> 5 then
183  ¦  ¦                                   DrawTextW(51,10 * i,1, YLabel);
184  ¦  E-------------------------------end;
185  ¦
186  ¦                              if Xexponent <> 0 then             {print X exponent on graph)
187  ¦  B-------------------------------begin
188  ¦  ¦                                DrawTextW(92,43,1,'x 10');
189  ¦  ¦                                DrawTextW(95,45,1,XexpLabel);
190  ¦  E-------------------------------end;
191  ¦
192  ¦                              if Yexponent <> 0 then              {print Y exponent on graph)
193  ¦  B-------------------------------begin
194  ¦  ¦                                DrawTextW(55,95,1,'x 10');
195  ¦  ¦                                DrawTextW(58,98,1,XexpLabel);
196  ¦  E-------------------------------end;
197  ¦
198  ¦                              DrawTextW(92,55,1,'REAL');  {label real/imag axes)
199  ¦                              DrawTextW(44,95,1,'IMAG');
200  ¦
201  ¦
202  ¦  E----------------------------Repeat until Keypressed;                (Put option menu on screen)
203  ¦                              quit:= false;
204  ¦  B----------------------------repeat
205  ¦  ¦                            if OpenLoop then Graph_Menu('Open Loop Nyquist',DumpGraph,quit)
206  ¦  ¦                                       else Graph_Menu('Closed Loop Nyquist',DumpGraph,quit);
207  ¦  ¦                            if DumpGraph then PrintGraphData;
208  ¦  E----------------------------until quit;
209  ¦                              LeaveGraphic;
210  E-------------------------end;
```

```
 1                              Procedure Root_Locus(G_eq:blocks);
 2                                 Label
 3                                   OnErr;
 4                                 Var
 5                                   I,J,code      : integer;
 6                                   PlotPole,
 7                                   PlotZero      : PlotArray;
 8                                   TempPoly,
 9                                   HoldPoly,
10                                   PlotRealPole,
11                                   PlotImagPole  : PolyArray;
12                                   DeltaGain,
13                                   StartGain,
14                                   EndGain,
15                                   Variable_Gain,
16                                   Xmin,Xmax,
17                                   Ymin,Ymax     : Real;
18                                   Neg_Feedback  : Boolean;
19                                   pg,qg         : real;
20                                   DumpGraph,quit : boolean;
21                                   list          : text;
22                                   LineCount     : integer;
23
24                                Procedure PrintGraphData;     {dumps rootlocus data to printer}
25  B-----------------------------Begin
26  |                               LeaveGraphic;
27  |                               Clrscr;
28  |                               Center('*** TURN ON PRINTER AND ALIGN PAPER ***',1,10,80);
29  |                               TextColor(green);
30  |                               msg('press <P> to Print, <F> to list to a file*, <Q> to quit print',1,13);
31  |                               TextColor(white);
32  |                               msg('*File option prints numbers to a file named "ROOTLOC.NUM"',1,17);
33  |                               msg('on the current drive.  Browse this off-line using DOS "type" command.',1,18);
34  |
35  |  B--------------------------repeat
36  |  |                             Read(kbd,ch);
37  |  |                               If (ch = 'F') or (ch = 'f') or (ch = 'P') or (ch = 'p') then
38  |  |  B--------------------------begin
39  |  |  |                             if (ch = 'F') or (ch = 'f') then
40  |  |  |  B--------------------------begin
41  |  |  |  |                             assign(list,'RootLoc.NUM');
42  |  |  |  |                             rewrite(list);
43  |  |  |  E--------------------------end
44  |  |  |                             else
45  |  |  |  B--------------------------begin
46  |  |  |  |                             assign(list,'LST:');
47  |  |  |  |                             rewrite(list);
48  |  |  |  E--------------------------end;
49  |  |  |
50  |  |  |                             LineCount := 0;
51  |  |  |                             writeln(list);
52  |  |  |                             writeln(list,'  ZEROS ');
53  |  |  |                             writeln(list);
54  |  |  |                             write(list,'                ');
55  |  |  |                             writeln(list,'    REAL                  IMAGINARY');
56  |  |  |                             writeln(list);  LineCount := LineCount + 6;
```

```
 57 | | |
 58 | | |                          with G_eq do
 59 | | |                          for i := 1 to NZeros do
 60 | | | B------------------------begin
 61 | | | |                          writeln(list,'                ',RealPartZero[i]:10:3,
 62 | | | |                                       '               ',ImagPartZero[i]:10:3);
 63 | | | |                          LineCount := LineCount + 1;
 64 | | | E------------------------end;
 65 | | |                          writeln(list); writeln(list);
 66 | | |                          writeln(list,'  POLES');
 67 | | |                          writeln(list);
 68 | | |                          write(list,'    K          ');
 69 | | |                          writeln(list,'    REAL                  IMAGINARY');
 70 | | |                          writeln(list); LineCount := LineCount + 7;
 71 | | |                          Variable_Gain := StartGain;
 72 | | |
 73 | | |                         {compute root locations for varying values of gain and print them}
 74 | | |
 75 | | |                          DeltaGain := (EndGain-StartGain)/100;pg:=0;qg:=0;
 76 | | |                          For J:= 1 to 100 do
 77 | | | B------------------------Begin
 78 | | | |                          Variable_Gain := Variable_Gain + DeltaGain;
 79 | | | |                          With G_eq do
 80 | | | |  B-----------------------Begin
 81 | | | |  |                           HoldPoly := DenCoeff;
 82 | | | |  |
 83 | | | |  |                           If Neg_Feedback then
 84 | | | |  |                               For I:= 1 to NZeros+1 do
 85 | | | |  |                                   HoldPoly[I] := HoldPoly[I] +(K*Variable_Gain *
 86 | | | |  |                                                                NumCoeff[I])
 87 | | | |  |                           else
 88 | | | |  |                               For I:= 1 to NZeros +1 do
 89 | | | |  |                                   HoldPoly[I] := HoldPoly[I] +(K*Variable_Gain *
 90 | | | |  |                                                                NumCoeff[I]);
 91 | | | |  |
 92 | | | |  |                           RootFinder(NPoles,HoldPoly,PlotRealPole,PlotImagPole,pg,qg);
 93 | | | |  |                           writeln(list,Variable_Gain:10:4); LineCount := LineCount + 1;
 94 | | | |  |                           for i := 1 to NPoles do
 95 | | | |  |  B--------------------------begin
 96 | | | |  |  |                            writeln(list,'            ',i:2,'     ',PlotRealPole[i]:10:3,
 97 | | | |  |  |                                                     ',PlotImagPole[i]:10:3);
 98 | | | |  |  |                            LineCount := LineCount + 1;
 99 | | | |  |  E--------------------------end;
100 | | | |  |                           writeln(list);LineCount := LineCount + 1;
101 | | | |  |                           if LineCount > 50 then
102 | | | |  |  B--------------------------begin
103 | | | |  |  |                            writeln(list, chr(12));
104 | | | |  |  |                            LineCount := 0;
105 | | | |  |  E--------------------------end;
106 | | | |  E-----------------------end;
107 | | | E--------------------------end;
108 | | E----------------------------end;
109 | E------------------------until ch in ['F','f','Q','q','p','P'];
110 |                            EnterGraphic;
111 |                            swapscreen;
112 |                            close(list);
113 E-------------------------end;
```

```
114
115                          {***************** END of PrintGraphData Procedure ************************}
116
117
118
119
120
121
122 B-------------------------Begin {Root_Locus Procedure}
123 ¦
124 ¦                               P[1]:= '5506N01001-010101';    {input handler driver strings}
125 ¦                               P[2]:= '5508N01002-010103';
126 ¦                               P[3]:= '1512N00503-010101';
127 ¦                               P[4]:= '1513N00504-010103';
128 ¦                               P[5]:= '1514N00505-010101';
129 ¦                               P[6]:= '1515N00506-010103';
130 ¦                               P[7]:= '4517A00207T010101';
131 ¦
132 ¦                               Clrscr; TextColor(white);
133 ¦                               Center('*** ROOT LOCUS PLOTTING ROUTINE ***',1,2,80);
134 ¦                               TextColor(Yellow);
135 ¦                               FillChar(s,100,#205);S:= copy(s,1,80);
136 ¦                               writeln;writeln(S);writeln;
137 ¦                               TextColor(green);
138 ¦                               writeln('What STARTING value for variable gain do you wish?:');
139 ¦                               writeln;
140 ¦                               writeln('What ENDING value for variable gain do you wish?:');
141 ¦                               TextColor(yellow);
142 ¦                               writeln; writeln(s);
143 ¦                               Center('*** VIEWING COORDINATES FOR ROOT LOCUS GRAPH ***',1,11,80);
144 ¦                               writeln; TextColor(green);
145 ¦                               writeln('X-Minimum: ');
146 ¦                               writeln('X-Maximum: ');
147 ¦                               writeln('Y-Minimum: ');
148 ¦                               writeln('Y-Maximum: ');
149 ¦                               writeln;writeln('Positive or Negative Feedback? (P or N):');
150 ¦
151 ¦                               Input_Handler('N0107',Escape);     {prompts for NEW inputs}
152 ¦                               writeln;writeln;
153 ¦                               writeln('Any changes to these parameters? (Y or N):');
154 ¦                               Input('A','',45,19,2,true,F1,F10);
155 ¦                               If answer='Y' then  Input_Handler('C0107',Escape); {prompts for changes}
156 ¦
157 ¦                               Val(filvar[1],StartGain,code);   {converts input strings into}
158 ¦                               Val(filvar[2],EndGain,code);     {numeric values}
159 ¦                               Val(filvar[3],Xmin,code);
160 ¦                               Val(filvar[4],Xmax,code);
161 ¦                               Val(filvar[5],Ymin,code);
162 ¦                               Val(filvar[6],Ymax,code);
163 ¦
164 ¦                               If copy(filvar[7],1,1) <> 'N' then Neg_feedback := false
165 ¦                                                             else Neg_feedback := true;
166 ¦
167 ¦                               INITGRAPHIC;      {define values for graphics routine}
168 ¦                               DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
169 ¦                               DefineWorld(1,XMin,YMin,XMax,YMax);
170 ¦                               SelectWindow(1);SelectWorld(1);
```

```
171 ¦                              DrawAxis(5,-5,0,0,0,0,0,0,false);
172 ¦
173 ¦                              {*************************************************************}
174 ¦                              {** Begin computation of closed loop roots and build plotting *}
175 ¦                              {** arrays.                                                   *}
176 ¦                              {*************************************************************}
177 ¦
178 ¦                              With G_eq do
179 ¦ B----------------------------Begin              {computes locations of zeros -- PlotZero[2,x]
180 ¦ ¦                                                 and PlotZero[3,x] are used because of a
181 ¦ ¦                                                 peculiarity in plotting routine that requires
182 ¦ ¦                                                 no less than 3 points to be plotted!         }
183 ¦ ¦                                  For 1:=1 to NZeros do
184 ¦ ¦ B--------------------------------Begin
185 ¦ ¦ ¦                                  PlotZero[1,1] := RealPartZero[1];
186 ¦ ¦ ¦                                  PlotZero[1,2] := ImagPartZero[1];
187 ¦ ¦ E--------------------------------end; {for}
188 ¦ ¦ B--------------------------------Case NZeros of
189 ¦ ¦ ¦                                  0 : ;
190 ¦ ¦ ¦ B--------------------------------1 : begin
191 ¦ ¦ ¦ ¦                                    PlotZero[2,1]:=PlotZero[1,1];
192 ¦ ¦ ¦ ¦                                    PlotZero[2,2]:=PlotZero[1,2];
193 ¦ ¦ ¦ ¦                                    PlotZero[3,1]:=PlotZero[1,1];
194 ¦ ¦ ¦ ¦                                    PlotZero[3,2]:=PlotZero[1,2];
195 ¦ ¦ ¦ ¦                                    DrawPolygon(PlotZero,1,-3,-3,3,0);
196 ¦ ¦ ¦ ¦                  {DrawPolygon is graphics routine to plot an array of points - PlotZero}
197 ¦ ¦ ¦ E--------------------------------end;
198 ¦ ¦ ¦ B--------------------------------2 : begin
199 ¦ ¦ ¦ ¦                                    PlotZero[3,1]:=PlotZero[1,1];
200 ¦ ¦ ¦ ¦                                    PlotZero[3,2]:=PlotZero[1,2];
201 ¦ ¦ ¦ ¦                                    DrawPolygon(PlotZero,1,-3,-3,3,0);
202 ¦ ¦ ¦ E--------------------------------end;
203 ¦ ¦ ¦                                else
204 ¦ ¦ ¦                                    DrawPolygon(PlotZero,1,NZeros,-3,3,0);
205 ¦ ¦ E--------------------------------end; {case}
206 ¦ E----------------------------end; {with}
207 ¦
208 ¦
209 ¦                          {** Begin computing values of closed loop roots with varying gain **}
210 ¦
211 ¦                              Variable_Gain := StartGain;
212 ¦                              DeltaGain := (EndGain-StartGain)/100;  {divide gain to plot 100 points}
213 ¦                              pg:=0;qg:=0;     {initial values for P and Q in root_finder procedure}
214 ¦
215 ¦                              For J:= 1 to 100 do  {calculate and plot 100 points per graph}
216 ¦ B----------------------------Begin
217 ¦ ¦                                Variable_Gain := Variable_Gain + DeltaGain;
218 ¦ ¦
219 ¦ ¦                                With G_eq do      {use G_equivalent block and make closed loop -
220 ¦ ¦                                                       with unity feedback}
221 ¦ ¦ B--------------------------------Begin
222 ¦ ¦ ¦                                  HoldPoly := DenCoeff;
223 ¦ ¦ ¦
224 ¦ ¦ ¦                                  If Neg_Feedback then
225 ¦ ¦ ¦                                    For 1:= 1 to NZeros+1 do
226 ¦ ¦ ¦                                      HoldPoly[I] := HoldPoly[I] +(K*Variable_Gain *
227 ¦ ¦ ¦                                                                    NumCoeff[I])
```

```
228  |  |  |                                  else
229  |  |  |                                     For I:= 1 to NZeros +1 do
230  |  |  |                                        HoldPoly[I] := HoldPoly[I] +(K*Variable_Gain *
231  |  |  |                                                                        NumCoeff[I]);
232  |  |  |
233  |  |  |                                     RootFinder(NPoles,HoldPoly,PlotRealPole,PlotImagPole,pg,qg);
234  |  |  |
235  |  |  |                                     For I:=1 to NPoles do      {fill plotting matrix with poles}
236  |  |  |  B----------------------------------Begin
237  |  |  |  |                                     PlotPole[I,1] := PlotRealPole[I];
238  |  |  |  |                                     PlotPole[I,2] := PlotImagPole[I];
239  |  |  |  E----------------------------------end;
240  |  |  |                                     AxisGlb := true;
241  |  |  |  B--------------------------Case NPoles of  {artificially fill plotting array if fewer
242  |  |  |  |                                                                        than 3 points}
243  |  |  |  |                                  0 : ;
244  |  |  |  |  B-------------------------------1 : begin
245  |  |  |  |  |                                  PlotPole[2,1]:=PlotPole[1,1];
246  |  |  |  |  |                                  PlotPole[2,2]:=PlotPole[1,2];
247  |  |  |  |  |                                  PlotPole[3,1]:=PlotPole[1,1];
248  |  |  |  |  |                                  PlotPole[3,2]:=PlotPole[1,2];
249  |  |  |  |  |                                  DrawPolygon(PlotPole,1,-3,-1,2,0);
250  |  |  |  |  E-------------------------------end;
251  |  |  |  |  B-------------------------------2 : begin
252  |  |  |  |  |                                  PlotPole[3,1]:=PlotPole[1,1];
253  |  |  |  |  |                                  PlotPole[3,2]:=PlotPole[1,2];
254  |  |  |  |  |                                  DrawPolygon(PlotPole,1,-3,-1,2,0);
255  |  |  |  |  E-------------------------------end;
256  |  |  |  |                                  else
257  |  |  |  |                                  DrawPolygon(PlotPole,1,-NPoles,-1,2,0);
258  |  |  |  E-----------------------------end;
259  |  |  E--------------------------end; {with}
260  |  E-------------------------end;   {root locus procedure}
261  |
262  |
263  |  E---------------------Repeat until KeyPressed;
264  |
265  |                        quit := false;
266  |  B---------------------repeat
267  |  |                      Graph_Menu('Root Locus',DumpGraph,quit);   {calls print/title menu}
268  |  |                      If DumpGraph then PrintGraphData;
269  |  E---------------------until quit;
270  |                        LeaveGraphic;
271  E---------------------end;
```

```
 1                          Program time_response;
 2                          {$I typedef.sys}
 3                          {$I graphix.sys}
 4                          {$I kernel.sys}
 5                          {$I windows.sys}
 6                          {$I axis.hgh}
 7                          {$I polygon.hgh}
 8                          {$I ut-mod01.inc}
 9                          {$I ut-mod02.inc}
10                          {$I ut-mod03.inc}
11                          {$I GrapMenu.inc}
12
13
14                          type
15                          BigMatrix = array[1..30,1..30] of real;
16                          BigVector = array[1..30] of real;
17
18                          var
19                           Psi,Phi,A,Atemp      : BigMatrix;
20                           temp, inputtype      : char;
21                           Offset,Slope,Tmax,
22                           RowSum,MaxRowSum,T,
23                           T1,OldMaxRowSum,
24                           Plottime,Uinput,PhiX,
25                           hold,Ymax,Ymin,TPlot,
26                           Amplitude, Freq,y    : real;
27                           Factorial,Plotindex,
28                           Nincr,code,i,
29                           j,l,m,n              : integer;
30                           DumpGraph,GoodNumbers,
31                           ClosedLoop,quit      : boolean;
32                           TRGEQ                : blocks;
33                           C,Xold,Xnext,Gamma   : BigVector;
34                           GraphArray,Inputarray: plotarray;
35                           List                 : text;
36
37
38
39                          Procedure PrintGraphData;     (dumps time-response data to printer)
40  B-----------------------Begin
41  |                         LeaveGraphic;
42  |                         Clrscr;
43  |                         Center('*** TURN ON PRINTER AND ALIGN PAPER ***',1,10,80);
44  |                         TextColor(green);
45  |                         msg('press <P> to Print, <F> to list to file * ,<Q> to quit print',1,13);
46  |                         TextColor(white);
47  |                         msg('* File option prints numbers to a file named "TIMERESP.NUM"',1,17);
48  |                         msg('on the current drive.  Browse the file off-line using the DOS "type" command.',1,18);
49  |
50  |  B--------------------repeat
51  |  |                       Read(kbd,ch);
52  |  |                         If (ch = 'F') or (ch = 'f') or (ch = 'p') or (ch = 'P') then
53  |  |  B--------------------begin
54  |  |  |                       if (ch = 'F') or (ch = 'f') then
55  |  |  |  B-----------------begin
56  |  |  |  |                       assign(list,'TimeResp.NUM');
```

```
 57 ¦ ¦ ¦ ¦                          rewrite(list);
 58 ¦ ¦ ¦ E-------------------------end
 59 ¦ ¦ ¦                          else
 60 ¦ ¦ ¦ B-------------------------begin
 61 ¦ ¦ ¦ ¦                          assign(list,'LST:');
 62 ¦ ¦ ¦ ¦                          rewrite(list);
 63 ¦ ¦ ¦ E-------------------------end;
 64 ¦ ¦ ¦
 65 ¦ ¦ ¦                        writeln(list, chr(78));   {skip over perforation in paper)
 66 ¦ ¦ ¦                        Writeln(list);
 67 ¦ ¦ ¦                        write(list);
 68 ¦ ¦ ¦                        writeln(list,'     TIME                Y (output)             U (input)');
 69 ¦ ¦ ¦                        writeln(list);
 70 ¦ ¦ ¦                        with G_eq do
 71 ¦ ¦ ¦                        for i := 1 to 200 do
 72 ¦ ¦ ¦                            writeln(list,'    ',GraphArray[i,1]:10:5,'               ',GraphArray[i,2]:12:4,
 73 ¦ ¦ ¦                                          '            ',InputArray[i,2]:12:4);
 74 ¦ ¦ E-------------------------end;
 75 ¦ E-----------------------until ch in ['F','f','Q','q','P','p'];
 76 ¦                          EnterGraphic;
 77 ¦                          swapscreen;
 78 ¦                          close(list);
 79 E-------------------------end;
 80
 81
 82
 83
 84
 85
 86                       Procedure Matrix_Mult(Matrix1,Matrix2:BigMatrix; var AnswerMatrix:BigMatrix;
 87                                                                              Order:integer);
 88                         var
 89                          i,j    : integer;
 90
 91 B-----------------------begin
 92 ¦                        for i:=1 to order do
 93 ¦                            for j:=1 to order do
 94 ¦                                AnswerMatrix[i,j] := 0;      {initialize the answer matrix}
 95 ¦
 96 ¦                        for i:= 1 to order do
 97 ¦                            for j:= 1 to order do
 98 ¦                                for L := 1 to order do
 99 ¦                                AnswerMatrix[i,j] := AnswerMatrix[i,j] + Matrix1[i,L]*Matrix2[L,j];
100 ¦
101 E-----------------------end;
102
103
104
105
106
107                        Procedure Scalar_Mult(Matrix1 : BigMatrix; scalar : real;
108                                                 var AnswerMatrix:BigMatrix;Order:integer);
109
110                          var i,j   : integer;
111
112 B-----------------------begin
113 ¦                          for i:= 1 to order do
```

```
114 |                                  for j:=1 to order do
115 |                                     AnswerMatrix[i,j]:= AnswerMatrix[i,j] * scalar;
116 E-------------------------------end;
117
118
119
120
121                              Procedure Matrix_Vector_Mult(Matrix1 :BigMatrix; Vector : BigVector;
122                                                          var AnswerVector:BigVector;Order:integer);
123
124                                 var i,j   : integer;
125
126 B-------------------------------begin
127 |
128 |                                 for i:= 1 to order do
129 |  B----------------------------------begin
130 |  |                                    hold:= 0;
131 |  |                                    for j:= 1 to order do
132 |  |                                       hold:= hold + Matrix1[i,j]*Vector[j];
133 |  |                                    AnswerVector[i]:= hold;
134 |  E----------------------------------end;
135 E-------------------------------end;
136
137
138
139
140
141 B---------------------------Begin
142 |                           initgraphic;leavegraphic;
143 |
144 |                           {*************** Prompt user for desired input/time limit ******************}
145 |
146 |
147 |                               clrscr;
148 |                               TextColor(white);
149 |                               center('*** Time Response Plotting Routine ***',1,2,80);
150 |                               fillchar(s,100,#205); S:= copy(s,1,80);
151 |                               writeln; writeln(s); writeln;
152 |                               TextColor(yellow);
153 |
154 |                               msg('What is the input to your system?  STEP (S) ',1,6);
155 |                               msg('                                   RAMP (R) ',1,7);
156 |                               msg('                                SIN WAVE (W) ',1,8);
157 |                               msg('                                 IMPULSE (I) ',1,9);
158 |
159 |  B----------------------------repeat
160 |  |                                Input('A','S',50,6,2,true,F1,F10);
161 |  |                                temp := copy(answer,1,1);
162 |  |                                if not (temp in ['S','R','W','I']) then beep(350,150);
163 |  E----------------------------until temp in ['S','R','W','I'];
164 |                               InputType := temp;
165 |
166 |                               msg('Input amplitude? ',1,11);
167 |                               Input('N','1',20,11,2,true,F1,F10);
168 |                               val(answer,Amplitude,code);
169 |
170 |  B----------------------------case InputType of
```

```
171  |  |
172  |  |  B-------------------------'R': begin
173  |  |  |                              msg('DC offset? ',1,13);
174  |  |  |                              Input('N','0',23,13,2,true,F1,F10);
175  |  |  |                              val(answer,Offset,code);
176  |  |  |                              msg('Slope? ', 25,13);
177  |  |  |                              Input('N','1',23,13,3,true,F1,F10);
178  |  |  |                              val(answer,slope,code);
179  |  |  E-----------------------------end;
180  |  |
181  |  |  B-------------------------'W': begin
182  |  |  |                              msg('Frequency? (rad/sec)',1,13);
183  |  |  |                              Input('N','',23,13,5,true,F1,F10);
184  |  |  |                              val(answer,Freq,code);
185  |  |  E-----------------------------end;
186  |  E-------------------------end;
187  |
188  |                              msg('Open (O) or Closed (C) Loop simulation? ',1,16);
189  |  B-------------------------repeat
190  |  |                            Input('A','C',45,16,2,true,F1,F10);
191  |  |                            temp := copy(answer,1,1);
192  |  |                            if not (temp in ['O','C']) then beep(350,150);
193  |  E-------------------------until temp in ['O','C'];
194  |                              if temp = 'C' then ClosedLoop:= true
195  |                                            else ClosedLoop:= false;
196  |
197  |
198  |                              msg('How many seconds of simulation would you like to see? (99 max)',1,20);
199  |  B-------------------------repeat
200  |  |                            Input('N','',65,20,5,true,F1,F10);
201  |  |                            val(answer,Tmax,code);
202  |  |                            if Tmax > 99 then beep(350,150);
203  |  E-------------------------until Tmax <= 99;
204  |
205  |                              clrscr;
206  |                              TextColor(white);
207  |                              center('*** Calculating the Time Response -- Please wait ***',1,10,80);
208  |                              TextColor(yellow);
209  |
210  |                          with G_eq do
211  |                           for i:= 1 to NPoles do
212  |                              DenCoeff[i] := DenCoeff[i]/ DenCoeff[NPoles+1];  {normalize polynomial}
213  |
214  |
215  |
216  |
217  |
218  |
219  |                         {************************ Make Closed Loop G equivalent **********************}
220  |
221  |                          if ClosedLoop then
222  |  B-------------------------begin
223  |  |                          with G_eq do          {compute closed-loop (unity feedback) G equivalent}
224  |  |  B----------------------begin                  {called TRGEQ -- which is also a blocks type record}
225  |  |  |
226  |  |  |                          TRGEQ.K := K;          {closed loop K same as open loop K}
227  |  |  |                          TRGEQ.NZEROS:=NZeros;  {as are the zeros of the function }
```

```
228 |  |  |
229 |  |  |                          for i:=1 to maxorder do TRGEQ.DenCoeff[i] := 0.0;   {initialize}
230 |  |  |
231 |  |  |
232 |  |  |                          for i:=1 to NZeros + 1 do
233 |  |  |  B-----------------------begin
234 |  |  |  |                         TRGEQ.DenCoeff[i] := NumCoeff[i] * K;   {C.L. denominator equals the }
235 |  |  |  |                         TRGEQ.NumCoeff[i] := NumCoeff[i];       {sum of open loop denominator}
236 |  |  |  E-----------------------end;                                       {and K times O.L. numerator  }
237 |  |  |
238 |  |  |                          for i:=1 to NPoles + 1 do TRGEQ.DenCoeff[i] := TRGEQ.DenCoeff[i] +
239 |  |  |                                                                         DenCoeff[i];
240 |  |  |
241 |  |  |
242 |  |  |                          if NPoles > NZeros then TRGEQ.NPOLES:=NPoles  {NPoles should always be}
243 |  |  |                                             else TRGEQ.NPOLES:=NZeros; {greater, but to be safe}
244 |  |  |
245 |  |  E-------------------------end; {with}
246 |  E----------------------------end
247 |                           else
248 |                            TRGEQ := G_eq;
249 |
250 |
251 |                          {******************** Fill the A-matrix ****************************}
252 |
253 |
254 |                                                    { A-matrix form:                }
255 |                          with TRGEQ do             {   | 0 1 0 0 ... 0 0 |         }
256 |  B-----------------------begin                      {   | 0 0 1 0 ... 0 0 |         }
257 |  |                                                 {   | 0 0 0 1 ... 0 0 |         }
258 |  |                                                 {   | ...             |         }
259 |  |                                                 {   | 0 0 0 0 ... 0 1 |         }
260 |  |                                                 {   | a b c d ... y z |         }
261 |  |                                  {where a,b, ... y,z are neg. coeff of denom poly}
262 |  |
263 |  |
264 |  |                            for i:= 1 to NPoles-1 do      {fill all but the bottom row}
265 |  |                                for j:=1 to NPoles do
266 |  |                                   if j = i+1 then A[i,j]:= 1
267 |  |                                              else A[i,j]:= 0;
268 |  |                            for j:= 1 to NPoles do
269 |  |                                A[NPoles,j] := -DenCoeff[j];
270 |  |
271 |  |                          {************************ Fill the C matrix ****************************}
272 |  |
273 |  |                             for i:= 1 to NPoles do
274 |  |  B--------------------------begin
275 |  |  |                             if i > NZeros + 1 then C[i]:= 0.0
276 |  |  |                                               else C[i]:= NumCoeff[i]*K;
277 |  |  |                             if NZeros = NPoles then C[i] := C[i] + K * NumCoeff[NZeros+1]*A[NPoles,i];
278 |  |  E--------------------------end;
279 |  |
280 |  |                          {******************** Select a sampling time interval T ********************}
281 |  |
282 |  |                            Nincr := 1000;
283 |  |                            T := (Tmax/Nincr);
284 |  |
```

```
285 | |
286 | |
287 | |                    {******************** Initialize Psi & Atemp  ******************************}
288 | |
289 | |                       Atemp := A;
290 | |                       Psi := A;                          {this will initialize psi to the value}
291 | |                       Scalar_Mult(Psi,T/2,Psi,NPoles);   {of the infinite series after the first}
292 | |                       for i:= 1 to NPoles do             {two terms I + A*T / 2!}
293 | |                          Psi[i,i]:=Psi[i,i] + 1.0;
294 | |
295 | |
296 | |                    {************ Compute more terms of the series & truncate ****************}
297 | |
298 | |                       Factorial := 2;  T1 := T;  Oldmaxrowsum:= 0.0;
299 | |
300 | | B---------------------repeat
301 | | | B-------------------begin
302 | | | |                     Factorial := Factorial * (i+1);  T1 := T1 * T;
303 | | | |                     Matrix_Mult(A,ATemp,Phi,NPoles);            {phi is used at temp}
304 | | | |                     ATemp:= Phi;                                {holding matrix to  }
305 | | | |                     Scalar_Mult(Phi,(T1/Factorial),Phi,NPoles); {large array }
306 | | | |                     for j:=1 to Npoles do
307 | | | |                         for m:= 1 to Npoles do
308 | | | |                             Psi[j,m]:=Psi[j,m] + Phi[j,m];
309 | | | |
310 | | | |
311 | | | |                    Maxrowsum:= 0.0;       {computes maxrowsum as measure of change in}
312 | | | |                    for j:= 1 to Npoles do             {last series term to be added.}
313 | | | | B------------------begin
314 | | | | |                   rowsum:= 0.0;
315 | | | | |                   for m:=1 to NPoles do
316 | | | | |                       rowsum:= rowsum + Psi[j,m];
317 | | | | |                   if rowsum > maxrowsum then maxrowsum := rowsum;
318 | | | | E------------------end;
319 | | | |                   if (abs(maxrowsum - oldmaxrowsum)/maxrowsum) < 0.001
320 | | | |                                                   then finished := false
321 | | | |                              {quit when .1%change} else finished := true;
322 | | | |
323 | | | |                   oldmaxrowsum := maxrowsum;
324 | | | E-------------------end;
325 | | E---------------------until Finished;
326 | |
327 | |                      Scalar_Mult(Psi,T,Psi,NPoles);
328 | |
329 | |
330 | |                    {******************** Calculate Phi matrix ******************************}
331 | |
332 | |                      Matrix_Mult(A,Psi,Phi,NPoles);
333 | |                      for i := 1 to NPoles do
334 | |                         Phi[i,i] := Phi[i,i] + 1.0;
335 | |
336 | |                    {********************* Calculate Gamma vector ****************************}
337 | |
338 | |                       for i:= 1 to NPoles do
339 | |                          Gamma[i] := Psi[i,NPoles];    {single input system with B vector:  }
340 | |                                                        {  B= [ 0 0 0 0 ... 0 1]  (transpose)}
341 | |
```

```
342 |  |
343 |  |                        {************** Compute the next state of x given x(0) = 0 ***************}
344 |  |
345 |  |
346 |  |                          Plottime := 0.0; PlotIndex := 1;  {initialize}
347 |  |                          for i:= 1 to NPoles do Xold[i]:= 0.0;    {init. prev. state}
348 |  |                          Ymax := 0.0; Ymin := 0.0;
349 |  |
350 |  |                          TextColor(white);
351 |  |                          gotoxy(15,10);
352 |  |                          write('*** Calculating        Points -- Please wait ***'); {countdown}
353 |  |                          TextColor(yellow);
354 |  |
355 |  |
356 |  |                          for N := 1 to Nincr do    {begin calculating next state and y}
357 |  |  B-----------------------begin
358 |  |  |                        GoToXY(32,10);  write(Nincr-N:5);   {display downcounter}
359 |  |  |
360 |  |  |                                {compute input at time Plottime}
361 |  |  |  B--------------------case Inputtype of
362 |  |  |  |                       'S' : Uinput := Amplitude;
363 |  |  |  |                       'R' : Uinput := Plottime * slope + offset;
364 |  |  |  |                       'I' : if plottime = 0 then Uinput := amplitude
365 |  |  |  |                                             else Uinput := 0.0;
366 |  |  |  |                       'W' : Uinput := Amplitude * sin(freq * plottime);
367 |  |  |  E--------------------end;
368 |  |  |
369 |  |  |
370 |  |  |                        Matrix_Vector_Mult(Phi,Xold,XNext,Npoles);  {compute new states}
371 |  |  |                        for i:= 1 to Npoles do
372 |  |  |                            Xnext[i]:= Xnext[i] + Gamma[i]*Uinput;
373 |  |  |
374 |  |  |
375 |  |  |                      {****************** Compute the value for the output y ********************}
376 |  |  |
377 |  |  |                           y:= 0.0;
378 |  |  |                           for i:= 1 to NPoles do
379 |  |  |                               if abs(y) < 1.0E07 then y:= y + C[i] * Xnext[i]
380 |  |  |                                                  else y:= 1.0E07;  {max y limit}
381 |  |  |
382 |  |  |                           if NZeros = NPoles then y := y + K * NumCoeff[NZeros+1] * Uinput;
383 |  |  |                           if y > Ymax then Ymax:= y;
384 |  |  |                           if y < Ymin then Ymin:= y;
385 |  |  |
386 |  |  |                           if N mod 5 = 0 then  {plot every 5th point}
387 |  |  |  B------------------------begin
388 |  |  |  |                           GraphArray[Plotindex,1] := Plottime;
389 |  |  |  |                           GraphArray[Plotindex,2] := y;
390 |  |  |  |                           InputArray[Plotindex,1] := Plottime;
391 |  |  |  |                           InputArray[Plotindex,2] := Uinput;
392 |  |  |  |                           Plotindex := Plotindex + 1;
393 |  |  |  E------------------------end;
394 |  |  |
395 |  |  |                           Plottime := Plottime + T; Xold := Xnext;
396 |  |  E-----------------------end;
397 |  |                           Ymax := 1.1 * Ymax;
398 |  |
```

```
399  ¦  ¦
400  ¦  ¦                              InitGraphic;
401  ¦  ¦                               DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
402  ¦  ¦                               DefineWorld(1,0,Ymax,tmax,Ymin);SelectWorld(1);
403  ¦  ¦                               SelectWindow(1); DrawBorder;
404  ¦  ¦                               DrawAxis(5,-5,0,0,0,0,0,0,false);
405  ¦  ¦                               DrawPolygon(GraphArray,1,-(Plotindex-1),0,0,0);
406  ¦  ¦                               DrawAxis(5,-5,0,0,0,0,0,0,false);
407  ¦  ¦                               SetLineStyle(1);  {dashed line for input signal}
408  ¦  ¦                               DrawPolygon(InputArray,1,-(Plotindex-1),0,0,0);
409  ¦  ¦                               SetLineStyle(0);
410  ¦  ¦
411  ¦  ¦  E----------------------repeat until keypressed;
412  ¦  ¦                           quit := false;
413  ¦  ¦  B----------------------repeat
414  ¦  ¦  ¦                          Graph_Menu('Time-Response',DumpGraph,quit);   {calls print/title menu}
415  ¦  ¦  ¦                          If DumpGraph then PrintGraphData;
416  ¦  ¦  E----------------------until quit;
417  ¦  ¦                           LeaveGraphic;
418  ¦  E------------------------end;
419  ¦                              assign(cadfile,'Cad.com');
420  ¦                              execute(cadfile);
421  ¦
422  E-------------------------end.
423
424
```

```
1   Program TwoParameterRootLocus;
2   {$I grapdef.sys}  {graphics type declarations}
3
4   const
5     maxorder = 20;  {maximum order for twoparameter system}
6
7   type
8     str80      = string[80];
9     stringarray = array[1..20] of str80;
10    PolyArray   = array[1..20] of real;
11
12  var
13    answer      : str80;
14    polish      : str80;
15    eval        : real;
16    InfixArray  : array[1..20] of str80;
17    DeltaStep,
18    Increm,
19    a,b         : real;
20    i,j,k,order : integer;
21    xmax,xmin,
22    ymax,ymin,
23    amax,amin,
24    bmax,bmin   : real;
25    DumpGraph,
26    Quit,Change,
27    StepA       : boolean;
28    EvalArray,
29    RealPart,
30    ImagPart    : PolyArray;
31    PlotPole    : PlotArray;
32    symbol      : integer;
33    hold        : string[10];
34    Line        : array[1..5] of string[14];
35    twochars    : string[2];
36    fivechars   : string[5];
37    CadFile     : file;
38
39  {$I graphix.sys}
40  {$I kernel.sys}
41  {$I windows.sys}
42  {$I polygon.hgh}
43  {$I axis.hgh}
44  {$I in-mod00.inc}    {slightly modified version of ut-mod00.inc}
45  {$I ut-mod01.inc}
46  {$I ut-mod02.inc}
47  {$I tp-mod03.inc}    {customized version of ut-mod03.inc}
48  {$I roots.inc}
49
50
51  Procedure Infix_to_Polish(Answer:str80; var RPN : str80); forward;
52  {forward declaration of proc to convert infix notation to reverse Polish}
53
54  Procedure Compute_Polish(Polish:str80;a,b:real; var evaluation:real); forward;
55  {forward declaration of the proc to evaluate the reverse Polish expression}
56
```

```
57
58                          {$I datadump.inc}     {program file to print numbers used to generate graph}
59                          {$I grapmenu.inc}     {graph menu options routine}
60
61
62
63
64
65                          {************************************************************************}
66                          {** The procedure Infix_to_Polish converts an algebraic expression (infix) **}
67                          {** to reverse Polish notation.  This conversion allows one-pass parsing  **}
68                          {** and evaluation of the expression which would otherwise not be possible.**}
69                          {************************************************************************}
70
71                          Procedure Infix_to_Polish;
72                           var
73                            Stack             : Array[1..50] of char;
74                            Top,P,I           : Integer;
75                            ch                : char;
76                            Firstchar,
77                            Prev_Digit        : Boolean;
78
79
80                             {*****************************************************************}
81                             {** The function Priority sets the heirarchy for operations    **}
82                             {** by setting the priority of an operator to be placed on a   **}
83                             {** stack.  The function is used by and internal to the proc   **}
84                             {** Infix_to_Polish.                                           **}
85                             {*****************************************************************}
86
87                                Function Priority(Ch:char): Integer;
88  B-----------------------------Begin
89  ¦  B--------------------------Case ch of
90  ¦  ¦                               '^'       : Priority:= 4;
91  ¦  ¦                               '*'       : Priority:= 3;
92  ¦  ¦                               '/'       : Priority:= 3;
93  ¦  ¦                               '+'       : Priority:= 2;
94  ¦  ¦                               '-'       : Priority:= 2;
95  ¦  ¦                               'a'..'b'  : Priority:= 1;
96  ¦  ¦                               'A'..'B'  : Priority:= 1;
97  ¦  ¦                               '0'..'9'  : Priority:= 1;
98  ¦  ¦                               '.'       : Priority:= 1;
99  ¦  ¦                               '('       : Priority:= 0;
100 ¦  ¦                               ')'       : Priority:= 0;
101 ¦  ¦                               ' '       : Priority:= 0;
102 ¦  E--------------------------end; {case}
103 E-----------------------------end; {function}
104
105
106 B---------------------------begin  {procedure Infix_to_Polish}
107 ¦                            RPN := ''; {output string}                    {initialization}
108 ¦                            Top := 0;  {operator stack pointer}
109 ¦                            FirstChar := True;     {FirstChar helps to find unary minus signs}
110 ¦                            Prev_Digit := False;   {Prev_Digit keeps spaces out of numeric constants}
111 ¦
112 ¦                            for i:= 1 to Length(Answer) do
113 ¦  B--------------------------begin
```

```
114                                ch := copy(Answer,i,1);  {scan the infix expression char by char)
115                                P:= Priority(ch);
116                                if Firstchar and (Ch = '-') then P:= 1;  {unary minus sign)
117
118                                if P = 1 then    {if the character is part of a constant, or variable)
119                             B------------------------------begin
120                                   if Prev_digit then       {suppresses blank spaces inside numeric)
121                                      RPN := concat(RPN,ch) {constants)
122                                   else
123                                      RPN := concat(RPN,' ',ch);
124                                   firstchar := False; Prev_Digit := True;
125                             E------------------------------end;
126                                if P > 1 then      {if an operator)
127                             B------------------------------begin
128                                     {checks priority of operator and arranges it on the stack)
129                                   while (Top > 0) and (Priority(Stack[top]) >= P) do
130                             B-----------------------------------begin
131                                      RPN := concat(RPN,' ',Stack[Top]); {if smaller priority, then)
132                                      Top := Top - 1;                    {operator placed on stack)
133                             E-----------------------------------end;                    {otherwise operator on top is placed )
134                                   Top := Top + 1;                {in output string and then current )
135                                   Stack[Top] := ch;               {operator is placed on stack)
136                                   Firstchar := True; Prev_Digit := False;
137                             E------------------------------end;
138
139                                 if ch = '(' then    {if opening paren then keeps inside stuff together)
140                             B------------------------------begin
141                                   Top := Top + 1;
142                                   Stack[Top] := '(';
143                                   FirstChar := True; Prev_Digit := False;
144                             E------------------------------end;
145                                if ch = ')' then     {closing paren causes inside stuff to be put on )
146                             B------------------------------begin            {output string together)
147                                   while Stack[Top] <> '(' do
148                             B-----------------------------------begin
149                                      RPN := concat(RPN,' ',Stack[Top]);
150                                      Top := Top - 1;
151                             E-----------------------------------end;
152                                   Top := Top - 1;       {skip over opening paren)
153                                   FirstChar := False; Prev_Digit := False;
154                             E------------------------------end;
155                          E---------------------------end;
156                                while Top > 0 do    {put remaining operators on output string)
157                          B-----------------------------------begin
158                                   RPN := concat(RPN,' ',Stack[Top]);
159                                   Top := Top - 1;
160                          E-----------------------------------end;
161
162
163                       E--------------------------end; {procedure infix_to_polish)
164
165
166
167
168
169
170                                {*************************************************************************)
```

```
171                              {*    Procedure Compute_Polish uses the string generated in Procedure     *}
172                              {*    Infix_to_Polish to evaluate the numeric expression.  The process    *}
173                              {*    uses a one-pass algorithm to analyze the expression and perform     *}
174                              {*    the appropriate stack or arithmetic operations.                     *}
175                              {*************************************************************************}
176
177
178                              Procedure Compute_Polish;
179
180                              var
181                               i,code       : integer;
182                               NumStack     : array[1..40] of real;
183                               Top          : integer;  {NumStack pointer}
184                               ch           : char;
185                               temp         : string[20];
186                               Value1,Value2,
187                               Value3       : real;
188
189
190                                        Procedure PUSH(Number:Real);     {push a number onto numeric stack}
191  B-------------------------------------begin
192  ¦                                        Top := Top + 1;
193  ¦                                        NumStack[Top] := Number;
194  E-------------------------------------end;
195
196                                        Procedure POP(var Number:Real);{pop a number off the numeric stack}
197  B-------------------------------------begin
198  ¦                                        Number:= NumStack[Top];
199  ¦                                        Top := Top -1;
200  E-------------------------------------end;
201
202                                        function Expon(Y,X:real):real;  {computes Y raised to X power}
203  B-------------------------------------Begin
204  ¦                                        Expon := exp( X * (ln(Y)));
205  E-------------------------------------end;
206
207
208
209
210  B-----------------------begin  {procedure Compute_Polish}
211  ¦                         temp := '';Top := 0;                {initialize}
212  ¦                         for i:= 1 to Length(polish) do {do one char at a time}
213  ¦  B--------------------------begin
214  ¦  ¦                          ch := copy(polish,i,1);    {get a character}
215  ¦  ¦  B-------------------------case ch of                                {and evaluate it}
216  ¦  ¦  ¦                          '0'..'9'  : temp := concat(temp,ch);   {real constant}
217  ¦  ¦  ¦                          '.'       : temp := concat(temp,ch);
218  ¦  ¦  ¦
219  ¦  ¦  ¦  B----------------------'-'       : begin
220  ¦  ¦  ¦  ¦                                   if copy(polish,i+1,1) <> ' ' then  {unary minus}
221  ¦  ¦  ¦  ¦                                      temp := concat(temp,ch)
222  ¦  ¦  ¦  ¦                                   else
223  ¦  ¦  ¦  ¦  B--------------------------------begin               {minus operator}
224  ¦  ¦  ¦  ¦  ¦                                   POP(Value1);
225  ¦  ¦  ¦  ¦  ¦                                   POP(Value2);
226  ¦  ¦  ¦  ¦  ¦                                   Value3 := Value2 - Value1;
227  ¦  ¦  ¦  ¦  ¦                                   PUSH(Value3);
```

```
228  ¦ ¦ ¦ ¦ E----------------------------------end;
229  ¦ ¦ ¦ E------------------------------------end;
230  ¦ ¦ ¦
231  ¦ ¦ ¦                          'a'      : PUSH(a);     {Put variables a and b onto numeric stack}
232  ¦ ¦ ¦                          'A'      : PUSH(a);
233  ¦ ¦ ¦                          'b'      : PUSH(b);
234  ¦ ¦ ¦                          'B'      : PUSH(b);
235  ¦ ¦ ¦
236  ¦ ¦ ¦ B------------------------'+'      : Begin        {add two numbers}
237  ¦ ¦ ¦ ¦                                   POP(Value1); POP(Value2);
238  ¦ ¦ ¦ ¦                                   Value3 := Value2 + Value1;
239  ¦ ¦ ¦ ¦                                   PUSH(Value3);
240  ¦ ¦ ¦ E------------------------------------end;
241  ¦ ¦ ¦
242  ¦ ¦ ¦ B------------------------'*'      : Begin        {multiply two numbers}
243  ¦ ¦ ¦ ¦                                   POP(Value1); POP(Value2);
244  ¦ ¦ ¦ ¦                                   Value3 := Value2 * Value1;
245  ¦ ¦ ¦ ¦                                   PUSH(Value3);
246  ¦ ¦ ¦ E------------------------------------end;
247  ¦ ¦ ¦
248  ¦ ¦ ¦ B------------------------'/'      : Begin        {divide two numbers}
249  ¦ ¦ ¦ ¦                                   POP(Value1); POP(Value2);
250  ¦ ¦ ¦ ¦                                   Value3 := Value2 / Value1;
251  ¦ ¦ ¦ ¦                                   PUSH(Value3);
252  ¦ ¦ ¦ E------------------------------------end;
253  ¦ ¦ ¦
254  ¦ ¦ ¦ B------------------------'^'      : Begin         {exponentiation}
255  ¦ ¦ ¦ ¦                                   POP(Value1); POP(Value2);
256  ¦ ¦ ¦ ¦                                   Value3 := Expon(Value2,Value1);
257  ¦ ¦ ¦ ¦                                   PUSH(Value3);
258  ¦ ¦ ¦ E------------------------------------end;
259  ¦ ¦ ¦
260  ¦ ¦ ¦ B------------------------' '      : Begin        {space is divider between numbers/operators}
261  ¦ ¦ ¦ ¦                                   if temp <> '' then
262  ¦ ¦ ¦ ¦ B----------------------------------begin
263  ¦ ¦ ¦ ¦ ¦                                    val(temp,Value1,code);
264  ¦ ¦ ¦ ¦ ¦                                    Push(Value1);
265  ¦ ¦ ¦ ¦ ¦                                    temp := '';
266  ¦ ¦ ¦ ¦ E----------------------------------end;
267  ¦ ¦ ¦ E------------------------------------end;
268  ¦ ¦ E--------------------------end; {case}
269  ¦ E----------------------------end; {for}
270  ¦
271  ¦                             if temp <> '' then       {lone constant in expression}
272  ¦ B---------------------------begin
273  ¦ ¦                              val(temp,Value1,code);
274  ¦ ¦                              PUSH(Value1);
275  ¦ E---------------------------end;
276  ¦
277  ¦                             POP(Value1);       {when finished answer will be top number on stack}
278  ¦                             Evaluation := Value1;
279  E--------------------------end; {procedure}
280
281
282
283
284                            {*****************************************************************************}
```

```
285                          {** Coeff_Input allows the user to input the algebraic expression which    **}
286                          {** describes the closed loop characteristic equation for the system with  **}
287                          {** up to two undetermined parameters called A and B.  The routine uses    **}
288                          {** standard algebraic, or infix, notation with parenthesis allowed.      **}
289                          {** Operators can include +, - , *, /, and ^ (exponentiation).  The unary  **}
290                          {** minus sign can be included in the expression.                          **}
291                          {**************************************************************************}
292
293                          Procedure Coeff_input;
294                          var
295                           code,i,linecount        : integer;
296                           validated               : boolean;
297
298
299  B-----------------------begin
300  |                        Clrscr; TextColor(white);
301  |                        Center('*** Two Parameter Root Locus ***',1,2,80);
302  |                        Fillchar(s,100,#205); S:= copy(s,1,80);TextColor(yellow);
303  |                        writeln;writeln(s);writeln;
304  |                        msg('What is the highest order coefficient in the Characteristic Equation?',1,8);
305  |                        msg('                                                          (20 maximum)',1,9);
306  |                        input('N','',78,8,2,false,F1,F10);
307  |                        val(answer,Order,code);
308  |
309  |                        clrscr;
310  |                        Center('*** Two Parameter Root Locus - Coefficient Input ***',1,1,80);
311  |                        writeln;writeln(s);
312  |
313  |                        P[1]:= '0805A07201T010101';
314  |                        P[2]:= '0807A07202T010102';
315  |                        P[3]:= '0809A07203T010103';
316  |                        P[4]:= '0811A07204T010104';
317  |                        P[5]:= '0813A07205T010105';
318  |                        P[6]:= '0815A07206T010106';
319  |                        P[7]:= '0817A07206T010107';
320  |                        P[8]:= '0819A07207T010108';
321  |                        P[9]:= '0821A07208T010109';
322  |                        P[10]:= '0823A07210T010110';
323  |                        P[11]:= '0805A07211T010111';
324  |                        P[12]:= '0807A07212T010112';
325  |                        P[13]:= '0809A07213T010113';
326  |                        P[14]:= '0811A07214T010114';
327  |                        P[15]:= '0813A07215T010115';
328  |                        P[16]:= '0815A07216T010116';
329  |                        P[17]:= '0817A07216T010117';
330  |                        P[18]:= '0819A07217T010118';
331  |                        P[19]:= '0821A07218T010119';
332  |                        P[20]:= '0823A07220T010120';
333  |
334  |
335  |                        for i:= 1 to order + 1 do       {writes prompts for coefficient input}
336  |  B--------------------------begin
337  |  |                           writeln('  ',order+1-i);
338  |  |                           writeln('s   = ');
339  |  E--------------------------end;
340  |                        str(order+1:2,twochars);
341  |                        if order+1 < 10 then            {computes invoking string for input_handler}
```

```
342 ¦ B----------------------------begin
343 ¦ ¦                              delete(twochars,1,1);
344 ¦ ¦                              insert('0',twochars,1);
345 ¦ E----------------------------end;
346 ¦
347 ¦                             if order + 1 > 10 then fivechars := 'N0110'
348 ¦                                               else fivechars := concat('N01',twochars);
349 ¦                             Input_handler(fivechars,escape);     {call the input_handler}
350 ¦
351 ¦                             if order + 1 > 10 then       {if greater than 10th order, write last 10 on next page}
352 ¦ B----------------------------begin
353 ¦ ¦                              fivechars := concat('N11',twochars);
354 ¦ ¦                              clrscr;
355 ¦ ¦                              Center('*** Two Parameter Root Locus - Coefficient Input ***',1,1,80);
356 ¦ ¦                              writeln;writeln(s); writeln;
357 ¦ ¦                              Input_Handler(fivechars,escape);
358 ¦ E----------------------------end;
359 ¦                              for i := 1 to order + 1 do
360 ¦                                  InfixArray[order+2-i] := Filvar[i];
361 E--------------------------end; {procedure Coeff_Input}
362
363
364
365
366
367                             {*****************************************************************************}
368                             {**  Select_Parameter_Range prompts the user for values of A and B to be    **}
369                             {**  used for plotting the root locus.  One parameter is stepped through    **}
370                             {**  five equal increments of range and the other is plotted "smoothly"     **}
371                             {**  by computing and plotting fifty values.                                **}
372                             {*****************************************************************************}
373
374                             Procedure Select_Parameter_Range;
375                              var
376                               code   : integer;
377
378 B---------------------------begin
379 ¦                              Clrscr; TextColor(white);
380 ¦                              Center('*** Parameter Selection Page ***',1,2,80);
381 ¦                              Fillchar(s,100,#205); S:= copy(s,1,80);TextColor(yellow);
382 ¦                              writeln;writeln(s);writeln;
383 ¦                              TextColor(green);     {print message explaining input for 2param proc}
384 ¦
385 ¦                              writeln('You will be varying the two parameters, A and B, through a range');
386 ¦                              writeln('of values you select.  You will also choose to STEP either A or B');
387 ¦                              writeln('which means that the chosen parameter''s range will be divided into');
388 ¦                              writeln('five (5) equal increments for plotting while the other parameter ');
389 ¦                              writeln('varies smoothly through its range.');
390 ¦                              writeln;writeln(s);writeln;writeln;
391 ¦
392 ¦                              P[1]:= '1516N01001-010103';
393 ¦                              P[2]:= '1517N01002-010101';
394 ¦                              P[3]:= '1519N01003-010103';
395 ¦                              P[4]:= '1520N01004-010101';
396 ¦                              P[5]:= '1522A00205T010101';
397 ¦
398 ¦                              TextColor(Yellow);              {prompt for user inputs of parameter values}
```

```
359  ¦                          writeln('A-minimum: ');
400  ¦                          writeln('A-maximum: ');
401  ¦                          writeln;
402  ¦                          writeln('B-minimum: ');
403  ¦                          writeln('B-maximum: ');
404  ¦                          writeln;writeln('Step A or B? :');
405  ¦
406  ¦                          Input_Handler('N0105',Escape);     (prompts for NEW inputs)
407  ¦
408  ¦  B-----------------------repeat
409  ¦  ¦                        ch := copy(filvar[05],1,1);
410  ¦  ¦                        if not(ch in ['A','B']) then
411  ¦  ¦                           Input_Handler('C0505',Escape);
412  ¦  E-----------------------until ch in ['A','B'];
413  ¦                          if ch = 'A' then stepA := true
414  ¦                                      else stepA := false;
415  ¦
416  ¦                          val(filvar[01],amin,code);
417  ¦                          val(filvar[02],amax,code);
418  ¦                          val(filvar[03],bmin,code);
419  ¦                          val(filvar[04],bmax,code);
420  ¦
421  E--------------------------end;
422
423
424
425                             {*************************************************************************}
426                             {**  Select_Window_Size prompts the user for the maximum and minimum values **}
427                             {**  of the plot in the x and y directions.                               **}
428                             {*************************************************************************}
429
430                             Procedure Select_Window_Size;
431
432                              var
433                               code   : integer;
434
435  B--------------------------begin
436  ¦                            Clrscr; TextColor(white);
437  ¦                            Center('*** Window Size Selection Page ***',1,2,80);
438  ¦                            Fillchar(s,100,#205); S:= copy(s,1,80);TextColor(yellow);
439  ¦                            writeln;writeln(s);writeln;writeln;writeln;
440  ¦
441  ¦                            P[6]:= '1508N01006-010103';
442  ¦                            P[7]:= '1510N01007-010101';
443  ¦                            P[8]:= '1513N01008-010103';
444  ¦                            P[9]:= '1515N01009-010101';
445  ¦
446  ¦                            writeln('X-minimum: ');     {prompts for window size;
447  ¦                            writeln;
448  ¦                            writeln('X-maximum: ');
449  ¦                            writeln;
450  ¦                            writeln;
451  ¦                            writeln('Y-minimum: ');
452  ¦                            writeln;
453  ¦                            writeln('Y-maximum: ');
454  ¦
455  ¦                            If change then Input_Handler('C0609',Escape)     {prompts for CHANGE inputs}
```

```
456 ¦                                   else Input_Handler('N0609',Escape);    {prompts for NEW   inputs}
457 ¦
458 ¦                              val(filvar[06],xmin,code);
459 ¦                              val(filvar[07],xmax,code);
460 ¦                              val(filvar[08],ymin,code);
461 ¦                              val(filvar[09],ymax,code);
462 ¦                              change := true;
463 ¦
464 E----------------------------end;
465
466
467
468
469
470                             {*************************************************************************}
471                             {**  Make_Legend draws a box on the screen and puts the graph legend inside **}
472                             {**  to alert the user which values of a and b are plotted using which      **}
473                             {**  symbol.  The box may be moved with cursor keys same as title box.      **}
474                             {*************************************************************************}
475
476                               Procedure Make_Legend;
477                                var
478                                 line1, line2, line3, line4, line5, line6   : string[20];
479
480 B----------------------------begin
481 ¦                               copyscreen;                          {make a window}
482 ¦                               SetLineStyle(0);
483 ¦                               DefineWindow(2,11,20,30,80);
484 ¦                               DefineWorld(2,0,30,28,0);
485 ¦                               SelectWorld(2); SelectWindow(2);
486 ¦                               DefineHeader(2,'Legend');
487 ¦                               SetBackground(0);
488 ¦                               SetHeaderOn;DrawBorder;
489 ¦                               Line1 := concat('   +++  ',line[1]);   {legend lines}
490 ¦                               Line2 := concat('   xxx  ',line[2]);
491 ¦                               Line3 := concat('   ***  ',line[3]);
492 ¦                               Line4 := concat('   OOO  ',line[4]);
493 ¦                               Line5 := concat('   YYY  ',line[5]);
494 ¦                               Line6 := '    Press F1 to Zoom';
495 ¦
496 ¦
497 ¦                               DrawTextW(1,4,1,Line1);               {write lines to window}
498 ¦                               DrawTextW(1,8,1,Line2);
499 ¦                               DrawTextW(1,12,1,Line3);
500 ¦                               DrawTextW(1,16,1,Line4);
501 ¦                               DrawTextW(1,20,1,Line5);
502 ¦                               DrawTextW(1,25,1,Line6);
503 ¦
504 ¦                               SetBreakOff; SetMessageOff;
505 ¦                               Set_Cap_Num(' ',' ',' '); Say_Cap_Num;
506 ¦
507 ¦  B-------------------------repeat                              {get user key input}
508 ¦  ¦                            read(kbd,ch);
509 ¦  ¦  B----------------------case ord(ch) of
510 ¦  ¦  ¦                          72 : MoveVer(-4,true); {up arrow}
511 ¦  ¦  ¦                          75 : MoveHor(-1,true); {left arrow}
512 ¦  ¦  ¦                          77 : MoveHor(1,true);  {right arrow}
```

```
513 ¦ ¦ ¦                          80 : MoveVer(4,true);  (down arrow)
514 ¦ ¦ ¦ B------------------------59 : begin quit := false;    (F1 pressed indicating that user wants)
515 ¦ ¦ ¦ ¦                              leavegraphic;     (to change graph scales)
516 ¦ ¦ ¦ E------------------------------end;
517 ¦ ¦ E--------------------------end;
518 ¦ E----------------------------until (ord(ch)= 13) or (ord(ch)=59);  (quit when <return> or <F1> pressed)
519 E------------------------------end;
520
521
522                            {#########################################################################}
523                            {##   Start of main program.  This part increments through the ranges of   ##}
524                            {##   A and B and plots the graph. The above procedures are called as      ##}
525                            {##   needed by the main program.                                          ##}
526                            {#########################################################################}
527
528 B--------------------------begin (Main Program)
529 ¦                          InitGraphic; LeaveGraphic;   (initialize graphics routines & screen)
530 ¦
531 ¦                          Select_Parameter_Range;  (prompt for A & B parameter ranges)
532 ¦
533 ¦
534 ¦
535 ¦                          if stepA then                    (set up steps and increments for A & B)
536 ¦ B------------------------begin
537 ¦ ¦                            DeltaStep := abs((amax-amin)/4);
538 ¦ ¦                            Increm    := abs((bmax-bmin)/50);
539 ¦ E------------------------end
540 ¦                          else
541 ¦ B------------------------begin
542 ¦ ¦                            DeltaStep := abs((bmax-bmin)/4);
543 ¦ ¦                            Increm    := abs((amax-amin)/50);
544 ¦ E------------------------end;
545 ¦
546 ¦
547 ¦                          Coeff_Input;          (input the coefficients of the characteristic equation)
548 ¦
549 ¦                          Change := false;      (initialize change boolean)
550 ¦
551 ¦ B----------------------Repeat
552 ¦ ¦                          Select_Window_Size;   (prompt for window size - x and y min's and max's)
553 ¦ ¦
554 ¦ ¦                          a:= amin; b:= bmin;   (initialize a and b)
555 ¦ ¦
556 ¦ ¦                          INITGRAPHIC;       (define values for graphics routine)
557 ¦ ¦                          DefineWindow(1,0,0,XMaxGlb,YMaxGlb);
558 ¦ ¦                          DefineWorld(1,xmin,ymax,xmax,ymin);
559 ¦ ¦                          SelectWindow(1);SelectWorld(1);
560 ¦ ¦                          DrawAxis(5,-5,0,0,0,0,0,0,false);
561 ¦ ¦
562 ¦ ¦
563 ¦ ¦                          for j := 1 to 5 do    (step through 5 values of selected stepping parameter)
564 ¦ ¦ B------------------------begin
565 ¦ ¦ ¦ B--------------------------case j of              (select which graph symbol will be used)
566 ¦ ¦ ¦ ¦                              1 : symbol := 1;  { + }
567 ¦ ¦ ¦ ¦                              2 : symbol := 2;  { x }
568 ¦ ¦ ¦ ¦                              3 : symbol := 7;  { * }
569 ¦ ¦ ¦ ¦                              4 : symbol := 8;  { o }
```

```
570 |  |  |  |                              5 : symbol := 6;  { Y }
571 |  |  |  E-------------------------end; {case}
572 |  |  |
573 |  |  |                           for k := 1 to 50 do   { smoothly move through the other parameter}
574 |  |  |  B------------------------begin                                    {range in 50 increments}
575 |  |  |  |
576 |  |  |  |                           for i:= 1 to Order + 1 do    {compute each coefficient value}
577 |  |  |  |  B---------------------------begin
578 |  |  |  |  |
579 |  |  |  |  |                              infix_to_polish(InfixArray[i],polish); {parse infix }
580 |  |  |  |  |
581 |  |  |  |  |                              Compute_Polish(polish,a,b,EvalArray[i]); {evaluate polish}
582 |  |  |  |  |
583 |  |  |  |  E---------------------------end;
584 |  |  |  |
585 |  |  |  |                           RootFinder(Order,EvalArray,RealPart,ImagPart,0,0); {find eq roots}
586 |  |  |  |
587 |  |  |  |                           For I:=1 to Order do       {fill plotting matrix with poles}
588 |  |  |  |  B---------------------------Begin
589 |  |  |  |  |                              PlotPole[I,1] := RealPart[I];
590 |  |  |  |  |                              PlotPole[I,2] := ImagPart[I];
591 |  |  |  |  E---------------------------end;
592 |  |  |  |                           AxisGlb := true;
593 |  |  |  |  B------------------------Case Order of  {artificially fill plotting array if fewer
594 |  |  |  |  |                                                              than 3 points}
595 |  |  |  |  |                             0 : ;                        {& plot the array for one value}
596 |  |  |  |  |  B----------------------1 : begin                              {of a and b}
597 |  |  |  |  |  |                          PlotPole[2,1]:=PlotPole[1,1];
598 |  |  |  |  |  |                          PlotPole[2,2]:=PlotPole[1,2];
599 |  |  |  |  |  |                          PlotPole[3,1]:=PlotPole[1,1];
600 |  |  |  |  |  |                          PlotPole[3,2]:=PlotPole[1,2];
601 |  |  |  |  |  |                          DrawPolygon(PlotPole,1,-3,-symbol,1,0);
602 |  |  |  |  |  E----------------------end;
603 |  |  |  |  |  B----------------------2 : begin
604 |  |  |  |  |  |                          PlotPole[3,1]:=PlotPole[1,1];
605 |  |  |  |  |  |                          PlotPole[3,2]:=PlotPole[1,2];
606 |  |  |  |  |  |                          DrawPolygon(PlotPole,1,-3,-symbol,1,0);
607 |  |  |  |  |  E----------------------end;
608 |  |  |  |  |                          else
609 |  |  |  |  |                          DrawPolygon(PlotPole,1,-Order,-symbol,1,0);
610 |  |  |  |  E------------------------end;
611 |  |  |  |
612 |  |  |  |                        if stepA then b := b + increm         {increment one parameter}
613 |  |  |  |                               else a := a + increm;
614 |  |  |  E-----------------------end;
615 |  |  |
616 |  |  |                        if stepA then
617 |  |  |  B---------------------begin
618 |  |  |  |                        str(a:10:4,hold);              {build legend string}
619 |  |  |  |                        Line[j] := concat('A = ',hold);
620 |  |  |  |                        a:= a + DeltaStep;
621 |  |  |  |                        b:= bmin;
622 |  |  |  E---------------------end  {next step}
623 |  |  |                        else
624 |  |  |  B---------------------begin
625 |  |  |  |                        str(b:10:4,hold);           {setup legend string}
626 |  |  |  |                        Line[j] := concat('B = ',hold);
```

```
627 | | | |                          b:= b + DeltaStep;
628 | | | |                          a:= amin;
629 | | | E--------------------------end;
630 | | |
631 | | E------------------------end;
632 | |                        quit := true;
633 | |                        Make_Legend;     {draw legend box on screen}
634 | E------------------------until quit;  {quit when finished with legend, repeat if user wants to }
635 |                                       {rescale graph by pressing F1 in Make_Legend routine}
636 |
637 |                        quit := false;  {this is a different use of quit - signalled when user}
638 |                                        {done with graph options menu}
639 | B------------------------repeat
640 | |                        Graph_Menu('2-Param.Root Locus',DumpGraph,quit);
641 | |                        If DumpGraph then PrintGraphData; {dump numbers to printer or file}
642 | E------------------------until quit;
643 |                        LeaveGraphic;
644 |
645 |                        Assign(CadFile,'Cad.COM');  {re-execute the main menu program when finished}
646 |                        Execute(CadFile);
647 E------------------------end.
```

```
                    (*********************************************************)
                    (*                                                       *)
                    (*              TURBO GRAPHIX version 1.05A              *)
                    (*                                                       *)
                    (*                Type definition module                 *)
                    (*                 Module version 1.00A                  *)
                    (*                                                       *)
                    (*                Copyright (C) 1985 by                  *)
                    (*                BORLAND International                  *)
                    (*                                                       *)
                    (*********************************************************)

                    const MaxWorldsGlb                =4;
                          MaxWindowsGlb               =16;
                          MaxPiesGlb                  =10;
                          MaxPlotGlb                  =500;
                          MaxOrder                    =9;
                          MaxBlocks                   =10;
                          StringSizeGlb               =80;
                          HeaderSizeGlb               =10;
                          RamScreenGlb:boolean        =true;
                          CharFile:string[StringSizeGlb]= '4x6.fon';
                          MaxFrocsGlb                 =27;
                          MaxErrsGlb                  =7;
                          Extension:string[4]         = '.blx';
                    type
                      wrkstring                       =string[StringSizeGlb];
B-------------------  WorldType                       =record
!                       x1,y1,x2,y2:real;
E-------------------------------------------------------end;
B-------------------  WindowType                      =record
!                       x1,y1,x2,y2:integer;
!                       header:wrkstring;
!                       drawn,top:boolean;
!                       size:integer;
E-------------------------------------------------------end;
                      worlds                          =array [1..MaxWorldsGlb] of WorldType;
                      windows                         =array [1..MaxWindowsGlb] of WindowType;
                      PlotArray                       =array [1..MaxPlotGlb,1..2] of real;
                      character                       =array [1..3] of byte;
                      CharArray                       =array [32..126] of character;
B-------------------  PieType                         =record
!                       area:real;
!                       text:wrkstring;
E-------------------------------------------------------end;
                      PieArray                        =array [1..MaxPiesGlb] of PieType;
                      BackgroundArray                 =array [0..7] of byte;
                      LineStyleArray                  =array [0..7] of boolean;
                      str255                          =  string[255];
                      str80                           =  string[80];
                      str40                           =  string[40];
                      str20                           =  string[20];
                      str8                            =  string[8];
                      str5                            =  string[5];
                      str4                            =  string[4];
                      str2                            =  string[2];
```

```
 57                          PolyArray                   =  array[1..MaxOrder]of Real;
 58  B-----------------------Blocks                      =  record
 59  |                                                      NZeros,NPoles: integer;
 60  |                                                      K           : real;
 61  |                                                      RealPartZero,
 62  |                                                      ImagPartZero,
 63  |                                                      RealPartPole,
 64  |                                                      ImagPartPole :PolyArray;
 65  |                                                      NumCoeff,
 66  |                                                      DenCoeff     :PolyArray;
 67  |                                                      LeadNumCoeff,
 68  |                                                      LeadDenCoeff :Real;
 69  |                                                      FeedBack,
 70  |                                                      Factored     :Boolean;
 71  E-------------------------------------------------------------end;
 72
 73                        var
 74                          X1WldGlb,X2WldGlb,Y1WldGlb,Y2WldGlb,AxGlb,AyGlb,BxGlb,ByGlb:real;
 75                          X1RefGlb,X2RefGlb,Y1RefGlb,Y2RefGlb                        :integer;
 76                          LinestyleGlb,MaxWorldGlb,MaxWindowGlb,WindowNdxGlb         :integer;
 77                          X1Glb,X2Glb,Y1Glb,Y2Glb                                    :integer;
 78                          XTextGlb,YTextGlb,VStepGlb                                 :integer;
 79                          PieGlb,DirectModeGlb,ClippingGlb,AxisGlb,HatchGlb          :boolean;
 80                          MessageGlb,ErkGlb,HeaderGlb,TopGlb,GrafModeGlb             :boolean;
 81                          CntGlb,ColorGlb                                            :byte;
 82                          ErrCodeGlb                                                 :byte;
 83                          LineStyleArrayGlb                                          :LineStyleArray;
 84                          ErrorProc                          :array [0..MaxProcsGlb] of ^WrkString;
 85                          ErrorCode                          :array [0..MaxErrsGlb] of ^WrkString;
 86                          PcGlb                              :string[40];
 87                          AspectGlb                          :real;
 88                          GrafBase                           :integer;
 89                          world                              :worlds;
 90                          window1                            :windows;
 91                          CharSet                            :CharArray;
 92                          Ch                                 : char;
 93                          Answer,Previous_Answer,S           : str80;
 94                          Template                           : str80;
 95                          P,Filvar                           : array[1..35] of str40; {menu Prompts}
 96                          InsertOn,Exit,
 97                          Escape,
 98                          F1,F10,
 99                          Use_Default,
100                          First_Run                          : Boolean;
101                          NBlocks                            : Integer;
102                          Block                              : Array[1..MaxBlocks] of Blocks;
103                          G_eq                               : Blocks;
104                          ConvergenceError                   : Boolean;
105                          Change                             : Boolean;
106                          Finished                           : Boolean;
107                          InputFile,CadFile,TimeFile,
108                          FreqFile, UtilFile, TwoParamFile   : File;
109                          Drive                              : str2;
110                          NegFeedBack                        : Boolean;
```

```
 1                        Procedure RootFinder (N:integer; var Coeff,RealPartRoot,ImagPartRoot: PolyArray; P1,Q1: Real);
 2
 3                                {*********************************************}
 4                                {* RootFinder solves for the roots of an     *}
 5                                {* arbitrary real polynomial by Bairstow's   *}
 6                                {* method.                                   *}
 7                                {*********************************************}
 8
 9                        Const
10                           Epsilon               = 0.00001;      {acceptable computation error}
11                        Var
12                           A,B,C                 : PolyArray;    {calculation arrays}
13                           P,Q,DeltaP,DeltaQ,Denom : Real;       {P,Q: coeff of quadratic factor}
14                                                                 {DeltaP,DeltaQ: iteration increm}
15                                                                 {Denom: Denom used to compute }
16                                                                 {       DeltaQ and DeltaP}
17                           IterationCount,i      : Integer;      {IterationCount: iteration counter}
18                                                                 {i: misc. loop counter}
19                           finished              : Boolean;      {finished: flag to tell when done}
20
21
22
23                         PROCEDURE Solve_Quadratic (var b,c,real1,imag1,real2,imag2:real);
24
25                                 {*****************************************}
26                                 {* Solve_Quadratic solves a polynomial of  *}
27                                 {* order 2 using the quadratic equation    *}
28                                 {*****************************************}
29
30                             Var
31                               S,radical : real;
32
33 B-----------------------------Begin                                {procedure solves quadratic equation)
34 |                             S:= -b/2.0;                          {of the form:                       }
35 |                             radical := (b*b)/4.0 - c;            {                                   }
36 |                             if radical > 0 then                  {       -b ± ( b} - 4ac             }
37 |  B--------------------------Begin          {real roots)          {            2a                     }
38 |  |                            real1 := S+sqrt(radical);         { with a = 1.                       }
39 |  |                            real2 := S-sqrt(radical);         {.....................................}
40 |  |                            imag1 := 0.0;
41 |  |                            imag2 := 0.0;
42 |  E--------------------------End
43 |                              else
44 |  B--------------------------Begin          {imag roots}
45 |  |                            real1 := S;
46 |  |                            real2 := S;
47 |  |                            imag1 := sqrt(-radical);
48 |  |                            imag2 := -sqrt(-radical);
49 |  E--------------------------End;
50 |
51 E-----------------------------End;
52
53 B-----------------------Begin   {PROCEDURE ROOTFINDER}
54 |
55 |                               {*********************************************}
56 |                               {* Fill A matrix with user's coefficients, most *}
```

```
57  |                                  {* significant coefficient in A[3], least signif*)
58  |                                  {* in A[n+3].  Initialize B and C matrices to 0.*)
59  |                                  {* ********************************************)
60  |
61  |                               For i:=3 to N+3 do A[i] := Coeff[N+4-i]/Coeff[N+1]; {normalize to unity
62  |                                                                                    highest order coeff}
63  |
64  |                               For i:=1 to maxorder do B[i] := 0.0;            {initialization)
65  |                               C:=B;  IterationCount:=1;  finished := false;
66  |                               P:=P1; Q:=Q1;      {best guesses for initial starting values - normally 0 )
67  |
68  |
69  |                             { This section solves simple polynomials of order zero or one.....)
70  |                             {.....}
71  |
72  |                               If N=0 then finished:= true;   {quit if polynomial is order 0}
73  |
74  |                                 If N=1 then        {solve polynomial of form:  x + a = 0 }
75  |  B------------------------------begin
76  |  |                              RealPartRoot[1]:= -A[4];
77  |  |                              ImagPartRoot[1]:=  0.0;
78  |  |                              Finished := True;
79  |  E------------------------------end;
80  |
81  |                                 If N=2 then     {solve polynomial of form:  x^2 + ax + b = 0  }
82  |  B------------------------------begin
83  |  |                              Solve_Quadratic(A[4],A[5],RealPartRoot[2],ImagPartRoot[2],RealPartRoot[1],ImagPartRoot[1]);
84  |  |                              Finished := True;
85  |  E------------------------------end;
86  |                             {.....}
87  |
88  |                                    {*****************************************************)
89  |                                    {* Bairstow's method searches iteratively for the     *)
90  |                                    {* quadratic factors of a given polynomial and uses   *)
91  |                                    {* that factor to reduce the order of the polynomial. *)
92  |                                    {* The process is continued until a polynomial of     *)
93  |                                    {* only order one or zero remains.                    *)
94  |                                    {*****************************************************)
95  |
96  |
97  |                               While (not finished) and (IterationCount < 40) do
98  |  B------------------------------Begin                      {solves iteratively for quadratic coeff)
99  |  |                              For i:= 3 to N+3 do        {P and Q of factor:  x} + Px +Q        )
100 |  |  B---------------------------Begin
101 |  |  |                              B[i} := A[i} - P*B[i-1] - Q*B[i-2];
102 |  |  |                              C[i] := B[i] - P*C[i-1] - Q*C[i-2];
103 |  |  E---------------------------end;
104 |  |
105 |  |                              Denom := C[n+1] * C[n+1} - (C[n+2}-B[n+2}) * C[n];
106 |  |                              If Denom <> 0.0 then
107 |  |  B---------------------------Begin
108 |  |  |                              DeltaP := (B[n+2]*C[n+1] - B[n+3]*c[n])/Denom;
109 |  |  |                              DeltaQ := (C[n+1]*B[N+3} - (C[n+2]-B[n+2])*B[n+2])/Denom;
110 |  |  |                              P:= P+DeltaP;  Q := Q+DeltaQ;
111 |  |  |                              If (abs(DeltaP) + abs(DeltaQ)) < epsilon then
112 |  |  |  B------------------------------Begin       { knowing P and Q solve the quadratic for roots)
113 |  |  |  |                              Solve_Quadratic(P,Q,RealPartRoot[n],ImagPartRoot[n],RealPartRoot[n-1],ImagPartRoot[n-1]);
```

```
 114  ¦  ¦  ¦  ¦                              n:=n-2;
 115  ¦  ¦  ¦  ¦
 116  ¦  ¦  ¦  ¦  B-------------------------case n of                    {if deflated polynomial is order}
 117  ¦  ¦  ¦  ¦  ¦                                0: finished := true;     { 0,1, or 2 solve here          }
 118  ¦  ¦  ¦  ¦  ¦  B-----------------------------1: Begin
 119  ¦  ¦  ¦  ¦  ¦  ¦                                  RealPartRoot[n] := -B[n+3]/B[n+2];
 120  ¦  ¦  ¦  ¦  ¦  ¦                                  ImagPartRoot[n] := 0.0;
 121  ¦  ¦  ¦  ¦  ¦  ¦                                  finished := true;
 122  ¦  ¦  ¦  ¦  ¦  E-----------------------------end;
 123  ¦  ¦  ¦  ¦  ¦  B-----------------------------2: Begin
 124  ¦  ¦  ¦  ¦  ¦  ¦                                  Solve_Quadratic(B[n+2],B[n+3],RealPartRoot[n],ImagPartRoot[n],RealPartRoot[n-1],ImagPartRo
ot[n-1
 125  ¦  ¦  ¦  ¦  ¦  ¦                                  finished := true;
 126  ¦  ¦  ¦  ¦  ¦  E-----------------------------end;
 127  ¦  ¦  ¦  ¦  ¦                             else
 128  ¦  ¦  ¦  ¦  ¦                                  For i:= 3 to N+3 do A[i] := B[i];  {B is reduced order
 129  ¦  ¦  ¦  ¦  ¦                                                                     polynomial.  Assign to
 130  ¦  ¦  ¦  ¦  ¦                                                                     A and iterate again   }
 131  ¦  ¦  ¦  ¦  ¦                                  IterationCount:=1;
 132  ¦  ¦  ¦  ¦  E-------------------------end; {case}
 133  ¦  ¦  ¦  E-----------------------------end
 134  ¦  ¦  ¦                           else
 135  ¦  ¦  ¦                              IterationCount := IterationCount+1;
 136  ¦  ¦  ¦
 137  ¦  ¦  E-------------------------end
 138  ¦  ¦                           else
 139  ¦  ¦  B-----------------------Begin
 140  ¦  ¦  ¦                            P:=P+1;  Q:=Q+1;  IterationCount:=1; beep(350,150);
 141  ¦  ¦  E-----------------------end;
 142  ¦  E-------------------------end;{while}
 143  E-----------------------end; {procedure}
```

```
 1                                   {*******************************************************}
 2                                   {**  The routine Expand_Poly takes real and complex   **}
 3                                   {**  conjugate factors of a polynomial and expands    **}
 4                                   {**  the factors into that polynomial.  The routine   **}
 5                                   {**  uses a shift-multiply-add algorithm for combining**}
 6                                   {**  the factors two at a time.  Complex conjugate    **}
 7                                   {**  factors are pre-multiplied in the procedure      **}
 8                                   {**  Conjug_Mult to yield a real second-order sub-    **}
 9                                   {**  polynomial and them combined with the remaining  **}
10                                   {**  factors.                                         **}
11                                   {*******************************************************}
12
13
14
15                              Procedure Expand_Poly(RealPt,ImagPt:PolyArray;var Poly:PolyArray;Order:integer);
16                                   type
17                                      Vec2 = Array[1..2] of real;
18                                   var
19                                      Temp      : PolyArray;
20                                      Hold      : Vec2;
21                                      I         : Integer;
22                                      Monomial  : Boolean;
23
24
25                                 Procedure Conjug_Mult(R,I:real;var Pol1,Pol2:real);
26  B------------------------------Begin
27  |                               Pol1 := I*I + R*R;     {local procedure to expand a quadratic factor}
28  |                               Pol2 := 2.0 * R;       {into a second-degree polynomial}
29  E------------------------------end;
30
31
32  B----------------------------Begin {Expand_Poly}
33  |
34  |
35  |                                   {*******************************************************}
36  |                                   {** Polynomial factors are loaded and multiplied.  If a *}
37  |                                   {** conjugate pair, then the pair is expanded first to  *}
38  |                                   {** a 2nd order polynomial then multiplied.             *}
39  |                                   {**                                                   *}
40  |                                   {**       (ax + b)(x + c) = ax} + (b+c)x + bc         *}
41  |                                   {*******************************************************}
42  |
43  |
44  |
45  |                               For I:=1 to MaxOrder do
46  |  B----------------------------Begin
47  |  |                               Temp[I]:=0.0; Poly[I]:=0.0;       {initialize the working arrays}
48  |  E----------------------------end;
49  |                               If Order = 0 then Poly[1] := 1.0;
50  |                               While Order > 0 do
51  |  B----------------------------begin
52  |  |                             If ImagPt[Order] = 0.0 then
53  |  |  B--------------------------begin                          {load initial real into arrays}
54  |  |  |                            Temp[1] := -RealPt[Order];
55  |  |  |                            Temp[2] := 1.0;
56  |  |  |                            Poly[1] := Temp[1];
```

```
 57 |  |  |                                  Poly[2] := Temp[2];
 58 |  |  |                                  Order   := Order -1;
 59 |  |  E--------------------------------end
 60 |  |                                   else                    {load converted conjugate into arrays}
 61 |  |  B--------------------------------begin
 62 |  |  |                                  Conjug_Mult(-RealPt[Order],ImagPt[Order],Temp[1],Temp[2]);
 63 |  |  |                                  Poly[1] := Temp[1];
 64 |  |  |                                  Poly[2] := Temp[2];
 65 |  |  |                                  Temp[3] := 1.0;
 66 |  |  |                                  Poly[3] := 1.0;
 67 |  |  |                                  Order   := Order -2;
 68 |  |  E--------------------------------end;
 69 |  |
 70 |  |                                 While Order > 0 do
 71 |  |  B------------------------------Begin
 72 |  |  |                                If ImagPt[Order] = 0.0 then
 73 |  |  |  B---------------------------Begin                    {load the next REAL factor}
 74 |  |  |  |                             Monomial := true;
 75 |  |  |  |                             Hold[1]  := -RealPt[Order];
 76 |  |  |  |                             Hold[2]  := 0.0;
 77 |  |  |  E---------------------------end
 78 |  |  |                               else
 79 |  |  |  B---------------------------Begin                     {load the next Conjugate pair}
 80 |  |  |  |                             Monomial := false;
 81 |  |  |  |                             Conjug_Mult(-RealPt[Order],ImagPt[Order],Hold[1],Hold[2]);
 82 |  |  |  E---------------------------end;
 83 |  |  |
 84 |  |  |                               If Not(Monomial) then {perform extra mult.required by 2nd order poly}
 85 |  |  |  B---------------------------Begin
 86 |  |  |  |                              For I:=MaxOrder downto 2 do Poly[I]:=Poly[I-1];
 87 |  |  |  |                              Poly[1] := 0.0;
 88 |  |  |  |                              For I:=1 to MaxOrder do Poly[I]:=Poly[I] + (Temp[I] * Hold[2]);
 89 |  |  |  |                              Order := Order - 1;
 90 |  |  |  E---------------------------end;
 91 |  |  |                                                         {process Real factor or continue
 92 |  |  |                                                          processing LSD of 2nd order factor}
 93 |  |  |
 94 |  |  |                               For I:= MaxOrder downto 2 do Poly[I] := Poly[I-1];
 95 |  |  |                               Poly[1] := 0.0;
 96 |  |  |
 97 |  |  |                               For I:=1 to MaxOrder do Poly[I]:=Poly[I] + (Temp[I] * Hold[1]);
 98 |  |  |
 99 |  |  |                               Temp := Poly;
100 |  |  |                               Order:= Order - 1;
101 |  |  E------------------------End {while}
102 |  E---------------------------End; {while}
103 |                              Temp := Poly;
104 |                              For I:= 1 to Order do Poly[Order+1-I]:= Temp[I];
105 E---------------------------End; {procedure}
106
```

```
 1                          Procedure Make_Geq;
 2
 3                          Var
 4                            i,j             : integer;    {general purpose loop indices)
 5                            Fwd_count_Z,
 6                            Fwd_count_P,
 7                            Fbk_count_Z,
 8                            Fbk_count_P      : integer;    {pole and zero counters)
 9                            RTemp, ITemp,
10                            TempPoly1,
11                            TempPoly2,
12                            Fwd_IZ,Fwd_RZ,
13                            Fwd_IP,Fwd_RP,
14                            Fbk_IZ,Fbk_RZ,
15                            Fbk_IP,Fbk_RP    : PolyArray;  {Geq factored form holding arrays)
16                            Fwd_gain,
17                            Fbk_gain         : Real;
18                            count,count2     : integer;    {temporary counters)
19                            FeedBackPathExists: Boolean;
20
21  B-------------------------Begin
22  ¦                         Fwd_count_Z:= 0;  Fwd_count_P:=0;   {initialize temporary holding variables)
23  ¦                         Fbk_count_Z:= 0;  Fbk_count_P:=0;
24  ¦                         Fwd_gain := 1;    Fbk_gain := 1;
25  ¦                         FeedBackPathExists := false;
26  ¦
27  ¦                         For j:=1 to NBlocks do        {repeat for every block in the loop)
28  ¦
29  ¦  B----------------------------Begin
30  ¦  ¦                           with Block[j] do
31  ¦  ¦  B-----------------------------begin
32  ¦  ¦  ¦                             If Feedback then       {if the block is in feedback path do this)
33  ¦  ¦  ¦  B--------------------------------begin
34  ¦  ¦  ¦  ¦                             For i:=1 to NZeros do
35  ¦  ¦  ¦  ¦  B-----------------------------------begin
36  ¦  ¦  ¦  ¦  ¦
37  ¦  ¦  ¦  ¦  ¦               {gather poles & zeros for all feedback blocks -- equivalent to multiplying
38  ¦  ¦  ¦  ¦  ¦                                                        block polynomials together)
39  ¦  ¦  ¦  ¦  ¦
40  ¦  ¦  ¦  ¦  ¦                                 Fbk_count_Z:=Fbk_count_Z + 1;
41  ¦  ¦  ¦  ¦  ¦                                 Fbk_RZ[Fbk_count_Z]:= RealPartZero[i];
42  ¦  ¦  ¦  ¦  ¦                                 Fbk_IZ[Fbk_count_Z]:= ImagPartZero[i];
43  ¦  ¦  ¦  ¦  E-----------------------------------end;
44  ¦  ¦  ¦  ¦                             For i:=1 to NPoles do
45  ¦  ¦  ¦  ¦  B-----------------------------------begin
46  ¦  ¦  ¦  ¦  ¦                                 Fbk_count_P:=Fbk_count_P + 1;
47  ¦  ¦  ¦  ¦  ¦                                 Fbk_RP[Fbk_count_P]:= RealPartPole[i];
48  ¦  ¦  ¦  ¦  ¦                                 Fbk_IP[Fbk_count_P]:= ImagPartPole[i];
49  ¦  ¦  ¦  ¦  E-----------------------------------end;
50  ¦  ¦  ¦  ¦                             Fbk_gain := Fbk_gain * K;
51  ¦  ¦  ¦  ¦                             FeedBackPathExists := true;
52  ¦  ¦  ¦  E--------------------------------end
53  ¦  ¦  ¦                              else
54  ¦  ¦  ¦  B--------------------------------begin
55  ¦  ¦  ¦  ¦
56  ¦  ¦  ¦  ¦               {gather poles and zeros for all forward path blocks -- same multiplicative
```

```
57  ¦  ¦  ¦  ¦                                                              effect)
58  ¦  ¦  ¦  ¦
59  ¦  ¦  ¦  ¦                                  For i:=1 to NZeros do
60  ¦  ¦  ¦  ¦  B-----------------------------------begin
61  ¦  ¦  ¦  ¦  ¦                                      Fwd_count_Z:=Fwd_count_Z + 1;
62  ¦  ¦  ¦  ¦  ¦                                      Fwd_RZ[Fwd_count_Z]:= RealPartZero[i];
63  ¦  ¦  ¦  ¦  ¦                                      Fwd_IZ[Fwd_count_Z]:= ImagPartZero[i];
64  ¦  ¦  ¦  ¦  E-----------------------------------end;
65  ¦  ¦  ¦  ¦
66  ¦  ¦  ¦  ¦                                  For i:=1 to NPoles do
67  ¦  ¦  ¦  ¦  B-----------------------------------begin
68  ¦  ¦  ¦  ¦  ¦                                      Fwd_count_P:=Fwd_count_P + 1;
69  ¦  ¦  ¦  ¦  ¦                                      Fwd_RP[Fwd_count_P]:= RealPartPole[i];
70  ¦  ¦  ¦  ¦  ¦                                      Fwd_IP[Fwd_count_P]:= ImagPartPole[i];
71  ¦  ¦  ¦  ¦  E-----------------------------------end;
72  ¦  ¦  ¦  ¦                                  Fwd_gain := Fwd_gain * K;
73  ¦  ¦  ¦  E--------------------------------end; {else}
74  ¦  ¦  ¦
75  ¦  ¦  E-----------------------------end; {with}
76  ¦  E--------------------------end; {for}
77  ¦
78  ¦
79  ¦                          {Make the G_equivalent block from poles and zeros gathered previously}
80  ¦
81  ¦                               with G_eq do
82  ¦  B----------------------------begin
83  ¦  ¦                               For i:=1 to MaxOrder do       {initialize G_eq factors}
84  ¦  ¦  B-----------------------------------begin
85  ¦  ¦  ¦                                RealPartZero[i]:=0.0; ImagPartZero[i]:=0.0;
86  ¦  ¦  ¦                                RealPartPole[i]:=0.0; ImagPartPole[i]:=0.0;
87  ¦  ¦  ¦                                NumCoeff[i]:=0.0;     DenCoeff[i]:=0.0;
88  ¦  ¦  E-----------------------------------end;
89  ¦  ¦                               NZeros:=0;
90  ¦  ¦                          .
91  ¦  ¦
92  ¦  ¦
93  ¦  ¦                          {******** G_equivalent Zeros are product of Forward path zeros and *********}
94  ¦  ¦                          {                    Feedback path poles.                              }
95  ¦  ¦
96  ¦  ¦
97  ¦  ¦                               For i:= 1 to Fwd_count_Z do
98  ¦  ¦  B-----------------------------------begin                  {collect forward path zeros}
99  ¦  ¦  ¦                             NZeros:=NZeros + 1;
100 ¦  ¦  ¦                             RealPartZero[NZeros] := Fwd_RZ[i];
101 ¦  ¦  ¦                             ImagPartZero[NZeros] := Fwd_IZ[i];
102 ¦  ¦  E-----------------------------------end;
103 ¦  ¦                               For i:= 1 to Fbk_count_P do
104 ¦  ¦  B-----------------------------------begin                  {collect feedback path poles}
105 ¦  ¦  ¦                             NZeros:=NZeros + 1;
106 ¦  ¦  ¦                             RealPartZero[NZeros] := Fbk_RP[i];
107 ¦  ¦  ¦                             ImagPartZero[NZeros] := Fbk_IP[i];
108 ¦  ¦  E-----------------------------------end;
109 ¦  ¦
110 ¦  ¦                               If NZeros = 0 then NumCoeff[1] := 1.0
111 ¦  ¦                                             else
112 ¦  ¦                                  Expand_Poly(RealPartZero, ImagPartZero, NumCoeff, NZeros);
113 ¦  ¦
```

```
114  ¦  ¦
115  ¦  ¦                        {******** G_equivalent Poles are product of Forward path zeros and *********}
116  ¦  ¦                        {****** Feedback path zeros added to the product of Forward Path poles *****}
117  ¦  ¦                        {********************** and feedback path poles. ***************************}
118  ¦  ¦
119  ¦  ¦                                  count:=0;  count2:=0;
120  ¦  ¦
121  ¦  ¦                                  For i:=1 to Fwd_count_Z do
122  ¦  ¦  B-----------------------------------begin                          {collect forward path zeros}
123  ¦  ¦  ¦                                      count:=count+1;
124  ¦  ¦  ¦                                      RTemp[count] := Fwd_RZ[i];
125  ¦  ¦  ¦                                      ITemp[count] := Fwd_IZ[i];              {and}
126  ¦  ¦  E-----------------------------------end;
127  ¦  ¦
128  ¦  ¦                                  For i:=1 to Fbk_count_Z do
129  ¦  ¦  B-----------------------------------begin                     {combine with feedback path zeros}
130  ¦  ¦  ¦                                      count:=count+1;
131  ¦  ¦  ¦                                      RTemp[count] := Fbk_RZ[i];
132  ¦  ¦  ¦                                      ITemp[count] := Fbk_IZ[i];
133  ¦  ¦  E-----------------------------------end;
134  ¦  ¦
135  ¦  ¦                                  Expand_Poly(Rtemp,Itemp, TempPoly1, count);
136  ¦  ¦
137  ¦  ¦                                  For i:=1 to count + 1 do    {multiply zeros polynomial by gains}
138  ¦  ¦                                      TempPoly1[i]:= TempPoly1[i] * Fwd_Gain * Fbk_Gain;
139  ¦  ¦
140  ¦  ¦
141  ¦  ¦
142  ¦  ¦                                  For i:=1 to Fwd_count_P do
143  ¦  ¦  B-----------------------------------begin                          {collect forward block poles}
144  ¦  ¦  ¦                                      count2:=count2+1;
145  ¦  ¦  ¦                                      RTemp[count2] := Fwd_RP[i];
146  ¦  ¦  ¦                                      ITemp[count2] := Fwd_IP[i];             {and}
147  ¦  ¦  E-----------------------------------end;
148  ¦  ¦
149  ¦  ¦                                  For i:=1 to Fbk_count_P do
150  ¦  ¦  B-----------------------------------begin                     {combine with feedback block poles}
151  ¦  ¦  ¦                                      count2:=count2+1;
152  ¦  ¦  ¦                                      RTemp[count2] := Fbk_RP[i];
153  ¦  ¦  ¦                                      ITemp[count2] := Fbk_IP[i];
154  ¦  ¦  E-----------------------------------end;
155  ¦  ¦
156  ¦  ¦                                  if not(FeedBackPathExists) then
157  ¦  ¦  B-----------------------------------begin                {if there are no feedback blocks, then}
158  ¦  ¦  ¦                                     for i:=1 to fwd_count_P do    {G_eq poles are fwd block poles}
159  ¦  ¦  ¦  B----------------------------------begin
160  ¦  ¦  ¦  ¦                                      RealPartPole[i] := Fwd_RP[i];
161  ¦  ¦  ¦  ¦                                      ImagPartPole[i] := Fwd_IP[i];
162  ¦  ¦  ¦  E----------------------------------end;
163  ¦  ¦  ¦                                  NPoles := Fwd_count_P;
164  ¦  ¦  ¦                                  Expand_Poly(RealPartPole,ImagPartPole,DenCoeff,NPoles);
165  ¦  ¦  E-----------------------------------end
166  ¦  ¦                                  else      {otherwise add two temporary polynomials}
167  ¦  ¦  B-----------------------------------begin
168  ¦  ¦  ¦                                  Expand_Poly(Rtemp,Itemp,TempPoly2, count2);
169  ¦  ¦  ¦
170  ¦  ¦  ¦                                  If count > count2 then NPoles := count    {order of largest}
```

```
171  ¦  ¦  ¦                                              else Npoles := count2;  {poly is NPoles}
172  ¦  ¦  ¦
173  ¦  ¦  ¦                                  For i:=1 to NPoles + 1 do
174  ¦  ¦  ¦                                      If NegFeedBack then DenCoeff[i]:=TempPoly2[i] + TempPoly1[i]
175  ¦  ¦  ¦                                                      else DenCoeff[i]:=TempPoly2[i] - TempPoly1[i];
176  ¦  ¦  ¦
177  ¦  ¦  ¦                                  RootFinder(NPoles,DenCoeff,RealPartPole,ImagPartPole,0,0);
178  ¦  ¦  E--------------------------------end;
179  ¦  ¦
180  ¦  ¦                                K := Fwd_gain;   {G_eq block gain is combined forward path gains}
181  ¦  ¦
182  ¦  ¦                                FeedBack := false;   {set G_equivalent boolean flags}
183  ¦  ¦                                Factored := true;
184  ¦  ¦                                LeadNumCoeff:= 1; LeadDenCoeff:= 1;
185  ¦  ¦
186  ¦  E-----------------------------end;  {with}
187  E--------------------------end; {procedure}
```

```
 1                          {*****************************************************************}
 2                          {**  ShowRoots is the procedure called in the main menu to      **}
 3                          {**  locate the closed loop roots of the loop equivalent xfer   **}
 4                          {**  function.                                                  **}
 5                          {*****************************************************************}
 6
 7                          Procedure ShowRoots(G_eq : Blocks);
 8                             Var
 9                              I,N,PosCounter:  Integer;
10                              RealRoot,
11                              ImagRoot,
12                              ClosedLoopPoly:  PolyArray;
13                              key,keyold    :  integer;
14                              not_erased    :  boolean;
15
16  B-----------------------Begin
17  ¦
18  ¦                        With G_eq do
19  ¦  B--------------------------Begin
20  ¦  ¦                           ClrScr; HighVideo;          {on-screen titles}
21  ¦  ¦                           Center('*** Block Transfer Function Closed-Loop Roots ***',1,2,80);
22  ¦  ¦                           writeln; writeln(s); writeln;
23  ¦  ¦                           writeln('ZEROS: ');
24  ¦  ¦
25  ¦  ¦                           For I:=1 to NZeros do        {position for output}
26  ¦  ¦  B------------------------begin
27  ¦  ¦  ¦                            PosCounter := (I mod 2) ;
28  ¦  ¦  ¦                            If PosCounter = 1 then writeln;
29  ¦  ¦  ¦
30  ¦  ¦  ¦                            LowVideo;      {write zeros - note: zeros of O.L. system same as
31  ¦  ¦  ¦                                            C.L. roots}
32  ¦  ¦  ¦                           write('s[',I,'] = ',RealPartZero[I]:10:3,' +j ',ImagPartZero[I]:10:3);
33  ¦  ¦  ¦                           write('            ');
34  ¦  ¦  E------------------------end;
35  ¦  ¦
36  ¦  ¦                           writeln; writeln; HighVideo;
37  ¦  ¦                           writeln('POLES: ');
38  ¦  ¦
39  ¦  ¦
40  ¦  ¦                                   {Make closed loop, unity-feedback system}
41  ¦  ¦                           For I:=1 to Maxorder do ClosedLoopPoly[I] := 0.0;
42  ¦  ¦
43  ¦  ¦                           For I:=1 to NZeros + 1 do ClosedLoopPoly[I] := NumCoeff[I] * K;
44  ¦  ¦
45  ¦  ¦                           For I:=1 to NPoles + 1 do ClosedLoopPoly[I] := ClosedLoopPoly[I] +
46  ¦  ¦                                                                          DenCoeff[I];
47  ¦  ¦
48  ¦  ¦                           If NPoles > NZeros then N := NPoles
49  ¦  ¦                                              else N := NZeros;
50  ¦  ¦
51  ¦  ¦                           RootFinder(N,ClosedLoopPoly,RealRoot,ImagRoot,0,0); {find factors}
52  ¦  ¦
53  ¦  ¦                           For I:=1 to N do
54  ¦  ¦  B------------------------begin                    {compute on-screen position}
55  ¦  ¦  ¦                            PosCounter := (I mod 2) ;
56  ¦  ¦  ¦                            If PosCounter = 1 then writeln;
```

```
57  ¦  ¦  ¦
58  ¦  ¦  ¦                              LowVideo;
59  ¦  ¦  ¦                                                    {output roots}
60  ¦  ¦  ¦                              write('s[',I,'] = ',RealRoot[I]:10:3,' +j ',ImagRoot[I]:10:3);
61  ¦  ¦  ¦                              write('             ');
62  ¦  ¦  E-----------------------------end;{for}
63  ¦  ¦
64  ¦  E--------------------------------end;{with}
65  ¦
66  ¦                                 HighVideo;
67  ¦                                 msg ('Press any key to continue or [Shift] [PrtSc] for hardcopy.',1,24);
68  ¦
69  ¦                                 {check keyboard buffer for value change.  If number changes by 1 or 2
70  ¦                                  indicates that shift key depressed.  If so, then remove "Press any key..."
71  ¦                                  prompt from screen so it won't print to printer}
72  ¦
73  ¦                                 keyold := mem[0000:1047]; not_erased := true;
74  ¦  B------------------------------Repeat
75  ¦  ¦                               key := mem[0000:1047];
76  ¦  ¦                               if ((key=keyold + 1) or (key=keyold + 2)) and (not_erased) then
77  ¦  ¦  B----------------------------begin
78  ¦  ¦  ¦                              GotoXY(1,24); write(' ':80);
79  ¦  ¦  ¦                              not_erased := false;
80  ¦  ¦  E----------------------------end;
81  ¦  E------------------------------Until KeyPressed;
82  ¦
83  E-----------------------------end;{Procedure ShowRoots}
```

```
 1                              Procedure Show_Factored_Roots;
 2                                 Var
 3                                  I,M,N,PosCounter:  Integer;
 4                                  NormFact        :  Real;
 5                                  key,keyold      :  Integer;
 6                                  not_erased      :  Boolean;
 7
 8  B---------------------------Begin
 9  ¦                            if NBlocks = 0 then    (print warning if no blocks currently in memory)
10  ¦  B--------------------------begin
11  ¦  ¦                           clrscr;TextColor(Red + Blink);
12  ¦  ¦                           center('WARNING',1,10,80); TextColor(White);
13  ¦  ¦                           center('*** There are NO blocks currently loaded ! ***',1,12,80);
14  ¦  E--------------------------end;
15  ¦
16  ¦                            For m:= 1 to NBlocks do   (for each block in the loop)
17  ¦  B--------------------------begin
18  ¦  ¦
19  ¦  ¦                           With Block[m] do
20  ¦  ¦  B-----------------------Begin             (print screen titles)
21  ¦  ¦  ¦                         ClrScr; HighVideo;
22  ¦  ¦  ¦                         Center('*** Block Transfer Function Roots ***',1,2,80);
23  ¦  ¦  ¦                         writeln; writeln(s);TextColor(white);
24  ¦  ¦  ¦                         NormFact:= LeadDenCoeff/LeadNumCoeff;
25  ¦  ¦  ¦                         write('BLOCK[',m,']');
26  ¦  ¦  ¦                         Writeln('                  Block Gain = ',K* NormFact:14:4);
27  ¦  ¦  ¦                         TextColor(green);
28  ¦  ¦  ¦                         writeln('ZEROS: ');
29  ¦  ¦  ¦
30  ¦  ¦  ¦                         For I:=1 to NZeros do    (write all zeros)
31  ¦  ¦  ¦  B--------------------------begin
32  ¦  ¦  ¦  ¦                           PosCounter := (I mod 2) ;  (write 2 across screen)
33  ¦  ¦  ¦  ¦                           If PosCounter = 1 then writeln;
34  ¦  ¦  ¦  ¦
35  ¦  ¦  ¦  ¦                           LowVideo;
36  ¦  ¦  ¦  ¦
37  ¦  ¦  ¦  ¦                          write('s[',I,'] = ',RealPartZero[I]:8:3,' +j ',ImagPartZero[I]:8:3);
38  ¦  ¦  ¦  ¦                          write('                ');
39  ¦  ¦  ¦  E--------------------------end;
40  ¦  ¦  ¦
41  ¦  ¦  ¦                         writeln; writeln; TextColor(green);
42  ¦  ¦  ¦                         writeln('POLES: ');
43  ¦  ¦  ¦
44  ¦  ¦  ¦                         For I:=1 to NPoles do   (repeat for poles)
45  ¦  ¦  ¦  B--------------------------begin
46  ¦  ¦  ¦  ¦                           PosCounter := (I mod 2) ;
47  ¦  ¦  ¦  ¦                           If PosCounter = 1 then writeln;
48  ¦  ¦  ¦  ¦
49  ¦  ¦  ¦  ¦                          LowVideo;
50  ¦  ¦  ¦  ¦
51  ¦  ¦  ¦  ¦                          write('s[',I,'] = ',RealPartPole[I]:8:3,' +j ',ImagPartPole[I]:8:3);
52  ¦  ¦  ¦  ¦                          write('                ');
53  ¦  ¦  ¦  E--------------------------end;(for)
54  ¦  ¦  ¦
55  ¦  ¦  E-----------------------end;(with)
56  ¦  ¦
```

```
                          HighVideo;
                          msg ('Press any key to continue or [Shift] [PrtSc] for hardcopy.',1,24);

                          {Test keyboard buffer for shift key depressed (shift key changes buffer
                           contents by 1 or 2).  If shift key is pressed, then a blank line is
                           written over the message "Press any key..." so it will not print}

                          keyold := mem[0000:1047];    {check initial value of buffer}
                          not_erased := true;
                  B-------Repeat
                           key := mem[0000:1047];  {repeatedly check buffer for changes}

                           if ((key=keyold + 1) or (key=keyold + 2)) and (not_erased) then
                             {if shift key pressed and message not previously erased then:}
                          B--begin
                             GotoXY(1,24); write(' ':80);    {overwrite message}
                             not_erased := false;
                          E--end;
                  E-------Until KeyPressed;   {if other key pressed then continue to next event}

          E---------end;{for}


                        With G_eq do     {repeat entire process for G-equivalent block}
                  B-----Begin
                          ClrScr; HighVideo;
                          Center('*** Loop Equivalent Block Transfer Function ***',1,2,80);
                          writeln; writeln(s);TextColor(white);
                          writeln('G equivalent Block                    Block Gain = ',K:14:4);
                          TextColor(green);
                          writeln('ZEROS: ');



                          For I:=1 to NZeros do
                  B-------begin
                             PosCounter := (I mod 2) ;
                             If PosCounter = 1 then writeln;

                             LowVideo;

                            write('s[',I,'] = ',RealPartZero[I]:8:3,' +j ',ImagPartZero[I]:8:3);
                            write('                ');
                  E-------end;

                          writeln; writeln; TextColor(green);
                          writeln('POLES: ');

                          For I:=1 to NPoles do
                  B-------begin
                             PosCounter := (I mod 2) ;
                             If PosCounter = 1 then writeln;

                            LowVideo;

                            write('s[',I,'] = ',RealPartPole[I]:8:3,' +j ',ImagPartPole[I]:8:3);
                            write('                ');
```

```
114  ¦  ¦  E--------------------------end;{for}
115  ¦  ¦
116  ¦  E-----------------------------end;{with}
117  ¦
118  ¦                              HighVideo;
119  ¦                              msg ('Press any key to continue or [Shift] [PrtSc] for hardcopy.',1,24);
120  ¦                              keyold := mem[0000:1047]; not_erased := true;
121  ¦  B--------------------------Repeat
122  ¦  ¦                           key := mem[0000:1047];
123  ¦  ¦                           if ((key=keyold + 1) or (key=keyold + 2)) and (not_erased) then
124  ¦  ¦  B------------------------begin
125  ¦  ¦  ¦                           GotoXY(1,24); write(' ':80);
126  ¦  ¦  ¦                           not_erased := false;
127  ¦  ¦  E------------------------end;
128  ¦  E--------------------------Until KeyPressed;
129  ¦
130  E-------------------------end;{Procedure ShowRoots}
```

```
                        {*************************************************************************}
                        {** Procedure ShowPoly is the utility routine to display the current loop **}
                        {** blocks in polynomial form.  It first shows each block, then the Geq   **}
                        {** block.                                                               **}
                        {*************************************************************************}

                        Procedure ShowPoly;
                           Var
                            VertPos,
                            HorizPos,
                            PosCounter,
                            I,J          :  Integer;
                            Exponent     :  String[2];
                            NormFact     :  Real;
                            Character    :  Char;
                            key,keyold   :  Integer;
                            not_erased   :  Boolean;


Begin
                          if NBlocks = 0 then   {print warning if no blocks in memory)
                            begin
                              clrscr;TextColor(Red + Blink);
                              center('WARNING',1,10,80); TextColor(White);
                              center('*** There are NO blocks currently loaded ! ***',1,12,80);
                            end;

                          For i:= 1 to NBlocks do   {print each block in loop)
                            begin
                             With Block[i] do
                              Begin
                                   ClrScr; HighVideo;   {screen title)
                                   Center('*** Coefficients of Block Polynomial ***',1,2,80);
                                   writeln; writeln(s); TextColor(Green);
                                   write('BLOCK #',i);
                                   {return normalized coefficients to state at input - won't )
                                   {confuse user)
                                   NormFact :=  LeadDenCoeff/LeadNumCoeff; TextColor(White);
                                   writeln('               Block Gain = ', K* NormFact:14:4);
                                   TextColor(Yellow);writeln('NUMERATOR: ');

                                   {compute screen positions for coefficient display)
                                   VertPos:= 8; LowVideo;
                                   For I:= NZeros+1 downto 1 do
                                     begin
                                       J:=NZeros+1 - I;
                                       PosCounter := (J mod 3)+1 ; {display 3 coeff across screen)
                                       HorizPos := PosCounter * 15;
                                       If PosCounter = 1 then VertPos := VertPos + 2;

                                       If I <> 1 then    {write "+ s" with proper exponent)
                                         begin
                                           msg('s  +',HorizPos,VertPos);
                                           str(I-1:2,Exponent);
```

```
 57 |  |  |  |  |                                  msg(Exponent,HorizPos+1,VertPos-1);
 58 |  |  |  |  E----------------------------------end;
 59 |  |  |  |                                  GoToXY(HorizPos-9,VertPos);
 60 |  |  |  |                                  write((LeadNumCoeff * NumCoeff[I]):8:3);
 61 |  |  |  E---------------------------------end;
 62 |  |  |
 63 |  |  |                                  HighVideo;writeln; writeln(s); writeln; {do same for denominator)
 64 |  |  |                                  writeln('DENOMINATOR: ');
 65 |  |  |
 66 |  |  |                                  VertPos := 16; LowVideo;
 67 |  |  |                                  For I:= NPoles+1 downto 1 do
 68 |  |  |  B-------------------------------begin
 69 |  |  |  |                                  J:=NPoles+1 - I;
 70 |  |  |  |                                  PosCounter := (J mod 4) + 1; HorizPos := PosCounter * 15;
 71 |  |  |  |                                  If PosCounter = 1 then VertPos := VertPos + 2;
 72 |  |  |  |
 73 |  |  |  |                                  If I <> 1 then
 74 |  |  |  |  B----------------------------------begin
 75 |  |  |  |  |                                  msg('s  +',HorizPos,VertPos);
 76 |  |  |  |  |                                  str(I-1:2,Exponent);
 77 |  |  |  |  |                                  msg(Exponent,HorizPos+1,VertPos-1);
 78 |  |  |  |  E----------------------------------end; {if)
 79 |  |  |  |                                  GoToXY(HorizPos-9,VertPos);
 80 |  |  |  |                                  write(LeadDenCoeff * DenCoeff[I]:8:3);
 81 |  |  |  E---------------------------------end; {for)
 82 |  |  |                                  msg ('Press any key to continue or [Shift] [PrtSc] for hardcopy.',1,24);
 83 |  |  |
 84 |  |  |                                  {check keyboard buffer for shift key (shift keys change buffer
 85 |  |  |                                  contents by 1 or 2). If shift key depressed, then the
 86 |  |  |                                  "Press any key..." message is erased from the screen so it
 87 |  |  |                                   won't print.)
 88 |  |  |
 89 |  |  |                                  keyold := mem[0000:1047]; {check initial condition of buffer)
 90 |  |  |                                  not_erased := true;  (initialize boolean)
 91 |  |  |  B-------------------------------Repeat
 92 |  |  |  |                                  key := mem[0000:1047]; {repeatedly check bufer)
 93 |  |  |  |                                  if ((key=keyold + 1) or (key=keyold + 2)) and (not_erased) then
 94 |  |  |  |                                     (if shift key depressed and the message has not already been erased)
 95 |  |  |  |  B----------------------------------begin
 96 |  |  |  |  |                                  GotoXY(1,24); write(' ':80); (write blank line across message)
 97 |  |  |  |  |                                  not_erased := false; {sets erased boolean)
 98 |  |  |  |  E----------------------------------end;
 99 |  |  |  |
100 |  |  |  E-------------------------------Until KeyPressed; {any other key will continue to next event)
101 |  |  |
102 |  |  E----------------------------------end; {with)
103 |  |
104 |  E-------------------------------end; {for)
105 |
106 |                                  With G_eq do   {repeat entire process with G_equivalent block)
107 |  B-------------------------------Begin
108 |  |                                  ClrScr; HighVideo;
109 |  |                                  Center('*** Coefficients of Block Polynomial ***',1,2,80);
110 |  |                                  writeln; writeln(s); TextColor(Green);
111 |  |                                  write('G-equivalent ');
112 |  |                                  NormFact :=  LeadDenCoeff/LeadNumCoeff; TextColor(White);
113 |  |                                  writeln('              Block Gain = ', K* NormFact:14:4);
```

```
114 ¦ ¦                                  TextColor(Yellow);writeln('NUMERATOR: ');
115 ¦ ¦
116 ¦ ¦                                  VertPos:= 8; LowVideo;
117 ¦ ¦                                  For I:= NZeros+1 downto 1 do
118 ¦ ¦ B--------------------------------begin
119 ¦ ¦ ¦                                    J:=NZeros+1 - I;
120 ¦ ¦ ¦                                    PosCounter := (J mod 3)+1 ; HorizPos := PosCounter * 15;
121 ¦ ¦ ¦                                    If PosCounter = 1 then VertPos := VertPos + 2;
122 ¦ ¦ ¦
123 ¦ ¦ ¦                                    If I <> 1 then
124 ¦ ¦ ¦ B----------------------------------begin
125 ¦ ¦ ¦ ¦                                      msg('s  +',HorizPos,VertPos);
126 ¦ ¦ ¦ ¦                                      str(I-1:2,Exponent);
127 ¦ ¦ ¦ ¦                                      msg(Exponent,HorizPos+1,VertPos-1);
128 ¦ ¦ ¦ E----------------------------------end;
129 ¦ ¦ ¦                                    GoToXY(HorizPos-9,VertPos);
130 ¦ ¦ ¦                                    write((LeadNumCoeff * NumCoeff[I]):8:3);
131 ¦ ¦ E--------------------------------end;
132 ¦ ¦
133 ¦ ¦                                  HighVideo;writeln; writeln(s); writeln;
134 ¦ ¦                                  writeln('DENOMINATOR: ');
135 ¦ ¦
136 ¦ ¦                                  VertPos := 16; LowVideo;
137 ¦ ¦                                  For I:= NPoles+1 downto 1 do
138 ¦ ¦ B--------------------------------begin
139 ¦ ¦ ¦                                    J:=NPoles+1 - I;
140 ¦ ¦ ¦                                    PosCounter := (J mod 4) + 1; HorizPos := PosCounter * 15;
141 ¦ ¦ ¦                                    If PosCounter = 1 then VertPos := VertPos + 2;
142 ¦ ¦ ¦
143 ¦ ¦ ¦                                    If I <> 1 then
144 ¦ ¦ ¦ B----------------------------------begin
145 ¦ ¦ ¦ ¦                                      msg('s  +',HorizPos,VertPos);
146 ¦ ¦ ¦ ¦                                      str(I-1:2,Exponent);
147 ¦ ¦ ¦ ¦                                      msg(Exponent,HorizPos+1,VertPos-1);
148 ¦ ¦ ¦ E----------------------------------end; {if}
149 ¦ ¦ ¦                                    GoToXY(HorizPos-9,VertPos);
150 ¦ ¦ ¦                                    write(LeadDenCoeff * DenCoeff[I]:8:3);
151 ¦ ¦ E--------------------------------end; {for}
152 ¦ ¦                                  msg ('Press any key to continue or [Shift] [PrtSc] for hardcopy.',1,24);
153 ¦ ¦                                  keyold := mem[0000:1047]; not_erased := true;
154 ¦ ¦ B--------------------------------Repeat
155 ¦ ¦ ¦                                   key := mem[0000:1047];
156 ¦ ¦ ¦                                   if ((key=keyold + 1) or (key=keyold + 2)) and (not_erased) then
157 ¦ ¦ ¦ B----------------------------------begin
158 ¦ ¦ ¦ ¦                                    GotoXY(1,24); write(' ':80);
159 ¦ ¦ ¦ ¦                                    not_erased := false;
160 ¦ ¦ ¦ E----------------------------------end;
161 ¦ ¦ E--------------------------------Until KeyPressed;
162 ¦ ¦
163 ¦ E------------------------------end; {with}
164 ¦
165 E----------------------------end;{Procedure ShowPoly}
```

```
 1                          {*************************************************************************}
 2                          {**  User_Polynomial_Roots is a procedure to allow the user to input any  **}
 3                          {**  arbitrary polynomial and automatically output the roots of that      **}
 4                          {**  polynomial.  It uses the standard input routines and RootFinder to   **}
 5                          {**  compute the roots of the polynomial.                                 **}
 6                          {*************************************************************************}
 7
 8                          Procedure User_Polynomial_Roots;
 9                            Var
10                              PolyOrder,
11                              PosCounter,
12                              VertPos,
13                              HorizPos,
14                              I,K,code      :  Integer;
15
16                              PolyCoeff,
17                              RealPart,
18                              ImagPart      :  PolyArray;
19
20                              Exponent      :  String[2];
21
22 B--------------------------Begin
23 |                            ClrScr; HighVideo;    {TITLE}
24 |                            Center('*** ROOTS OF A USER INPUT POLYNOMIAL ***',1,1,80);
25 |                            Fillchar(s,100,#205); s:= copy(s,1,80);
26 |                            writeln; writeln(s); writeln;
27 |                            writeln('Polynomial Coefficient Input: ');
28 |                            writeln;
29 |
30 |                            msg('What is the Order of the Polynomial? ',10,8); {prompt user for poly}
31 |                            input('N','',60,8,2,true,F1,F10);
32 |                            val(Answer,PolyOrder,code);
33 |                            writeln;
34 |
35 |                            VertPos:= 12;              {position prompts on screen}
36 |                            For I:= PolyOrder+1 downto 1 do
37 |  B-------------------------begin
38 |  |                             K:=PolyOrder+1 - I;
39 |  |                             PosCounter := (K mod 6) + 1; HorizPos := PosCounter * 10;
40 |  |                             If PosCounter = 1 then VertPos := VertPos + 2;
41 |  |
42 |  |                             If I <> 1 then
43 |  |  B----------------------------begin
44 |  |  |                             msg('x  +',HorizPos,VertPos);
45 |  |  |                             str(I-1:2,Exponent);
46 |  |  |                             msg(Exponent,HorizPos+1,VertPos-1);
47 |  |  E----------------------------end;
48 |  |                             input('N','',HorizPos-5,VertPos,5,true,F1,F10);
49 |  |                             val(answer,PolyCoeff[I],code);
50 |  E-------------------------end;
51 |
52 |                            ClrScr; Center('*** Computing Roots -- Please Wait ***',1,10,80); {wait msg}
53 |
54 |                            RootFinder(PolyOrder,PolyCoeff,RealPart,ImagPart,0,0);  {solve for roots}
55 |
56 |                            ClrScr; HighVideo; Beep(350,150);  {alert user when finished}
```

```
57  ¦
58  ¦
59  ¦                          Center('*** Roots of the Polynomial ***',1,2,80);   (root display)
60  ¦                          writeln; writeln(s); writeln;
61  ¦
62  ¦                          For I:=1 to PolyOrder do     (compute screen positions)
63  ¦  B-----------------------begin
64  ¦  ¦                           PosCounter := (I mod 2) ;
65  ¦  ¦                           If PosCounter = 1 then
66  ¦  ¦  B--------------------------begin
67  ¦  ¦  ¦                           writeln; writeln;
68  ¦  ¦  E--------------------------end;                    (display the roots)
69  ¦  ¦                           LowVideo;
70  ¦  ¦                           write('s[',I,'] = ',RealPart[I]:8:3,' +j ',ImagPart[I]:8:3);
71  ¦  ¦                           write('                ');
72  ¦  E-----------------------end;
73  ¦                          HighVideo;
74  ¦                          msg ('Press any key to continue or [Shift] [PrtSc] for hardcopy.',1,24);
75  ¦  E--------------------REPEAT UNTIL KEYPRESSED;
76  E-------------------end;
```

```
 1                          {*****************************************************************************}
 2                          {**  Graph_Menu provides a window on screen and offers the user options to **}
 3                          {**  make a title, print the graph, print the numbers from the graph, or   **}
 4                          {**  quit and return to the menu.                                          **}
 5                          {*****************************************************************************}
 6
 7                          Procedure Graph_Menu(TitleWindowName:Str20;var DumpGraphData,quit : boolean);
 8                           var
 9                            Line1, Line2, Line3  : string[40];  (graph prompt lines)
10
11
12                           Procedure TitlePrompt;    (Prompts for user supplied graph title)
13  B------------------------begin
14  |                          TextColor(White);
15  |                          Center('*** Make Graph Title ***',1,2,80);
16  |
17  |                          P[1]:= '1010A04001-010100';
18  |                          P[2]:= '1012A04002-010100';
19  |                          P[3]:= '1014A04003-010100';
20  |
21  |                          textcolor(yellow);
22  |                          msg('Line #1:',1,10);
23  |                          msg('Line #2:',1,12);
24  |                          msg('Line #3:',1,14);
25  |
26  |                          textcolor(green);
27  |                          msg('Type in the title you wish for your graph.',6,20);
28  |
29  |                          Input_handler('N0103',escape);
30  |
31  |                          Line1:= copy(filvar[1],1,40);
32  |                          Line2:= copy(filvar[2],1,40);
33  |                          Line3:= copy(filvar[3],1,40);
34  |
35  E------------------------end;
36
37
38                           Procedure ShowTitle;    (makes title block and writes title to block)
39
40  B------------------------begin
41  |                          copyscreen;
42  |                          SetLineStyle(0);
43  |                          DefineWindow(3,11,20,40,60);
44  |                          DefineWorld(3,0,16,40,0);
45  |                          SelectWorld(3); SelectWindow(3);
46  |                          DefineHeader(3,TitleWindowName); (puts header on box)
47  |                          SetBackground(0);
48  |                          SetHeaderOn;DrawBorder;
49  |                          DrawTextW(1,4,1,Line1);   (draw title supplied by user)
50  |                          DrawTextW(1,8,1,Line2);
51  |                          DrawTextW(1,12,1,Line3);
52  |                          SetBreakOff; SetMessageOff;
53  |
54  |  B---------------------repeat
55  |  |                        read(kbd,ch);
56  |  |  B------------------case ord(ch) of      (allow user to move title box anywhere on screen)
```

```
57  ¦  ¦  ¦                          72 : MoveVer(-4,true); {up arrow}
58  ¦  ¦  ¦                          75 : MoveHor(-1,true); {left arrow}
59  ¦  ¦  ¦                          77 : MoveHor(1,true);  {right arrow}
60  ¦  ¦  ¦                          80 : MoveVer(4,true);  {down arrow}
61  ¦  ¦  E-----------------------end;
62  ¦  E--------------------------until ord(ch)= 13;  {freeze box and continue with <return> key}
63  E-----------------------------end;
64
65
66
67  B-------------------------begin
68  ¦                           DumpGraphData:= False;
69  ¦                           copyscreen; SetLineStyle(0);     {save underlying screen and display menu box}
70  ¦                           DefineWindow(4,11,20,35,90);
71  ¦                           DefineWorld(4,0,20,20,0);
72  ¦                           SelectWorld(4); SelectWindow(4);
73  ¦                           DefineHeader(4, 'Graph Options Menu');
74  ¦                           SetHeaderOn; SetBackground(0);DrawBorder;
75  ¦                           DrawTextW(1,4,1,'<P>  Print Graph');   {display menu options}
76  ¦                           DrawTextW(1,7,1,'<T>  Make Title Block');
77  ¦                           DrawTextW(1,10,1,'<N>  Print Table of Numbers');
78  ¦                           DrawTextW(1,13,1,'     used to Generate Graph');
79  ¦                           DrawTextW(1,17,1,'<Q> Return to Main Menu>');
80  ¦  B----------------------repeat
81  ¦  ¦                         Option;
82  ¦  ¦  B-------------------case ch of    {interpret user input}
83  ¦  ¦  ¦  B-----------------'P': begin
84  ¦  ¦  ¦  ¦                        swapscreen;  {redisplays screen without menu box}
85  ¦  ¦  ¦  ¦                        hardcopy(false,1);  {print to printer}
86  ¦  ¦  ¦  ¦                        swapscreen;   {brings back screen with menu box}
87  ¦  ¦  ¦  ¦                        ch := 'P';
88  ¦  ¦  ¦  E-------------------------end;
89  ¦  ¦  ¦  B-----------------'T': begin
90  ¦  ¦  ¦  ¦                        leavegraphic;   {leave graphics screen}
91  ¦  ¦  ¦  ¦                        TitlePrompt;    {prompt for title}
92  ¦  ¦  ¦  ¦                        entergraphic;   {return to graphics mode}
93  ¦  ¦  ¦  ¦                        swapscreen;     {bring back graph}
94  ¦  ¦  ¦  ¦                        ShowTitle;      {display title box on screen}
95  ¦  ¦  ¦  ¦                        copyscreen;     {save graph with title box}
96  ¦  ¦  ¦  ¦                        ch := 'T';
97  ¦  ¦  ¦  E-------------------------end;
98  ¦  ¦  ¦  B-----------------'N': begin
99  ¦  ¦  ¦  ¦                        DumpGraphData := True;  {sets boolean to cause numbers to be printed}
100 ¦  ¦  ¦  ¦                        swapscreen;             {bring back graphics screen}
101 ¦  ¦  ¦  ¦                        ch := 'N';
102 ¦  ¦  ¦  E-------------------------end;
103 ¦  ¦  ¦                       'Q': ;
104 ¦  ¦  ¦
105 ¦  ¦  E--------------------end;
106 ¦  E-----------------------until ch in ['P','T','N','Q'];
107 ¦                         if ch = 'Q' then Quit := true
108 ¦                                     else Quit := false;
109 E-------------------------end;
110
111
```

```
 1                          Procedure PrintGraphData;     {dumps rootlocus data to printer}
 2
 3                            var
 4                              LineCount,i,j,k,l   : integer;
 5                              FirstPage           : boolean;
 6                              List                : text;
 7
 8                            Procedure NewPage;
 9  B-------------------------begin
10  ¦                           if not(FirstPage) then
11  ¦                              Write(list,chr(12));   {formfeed for all but first page}
12  ¦
13  ¦                           writeln(list,'  POLE LOCATIONS');       {write titles}
14  ¦                           if StepA then writeln(list,'A = ',amin:10:3)
15  ¦                                   else writeln(list,'B = ',bmin:10:3);
16  ¦                           if StepA then write(list,'        B      Pole')
17  ¦                                   else write(list,'        A      Pole');
18  ¦                           writeln(list,' #    REAL                IMAGINARY');
19  ¦                           for l := 1 to 60 do write(list,'_'); {draw a line across page}
20  ¦                           writeln(list);
21  ¦                           FirstPage := False;
22  ¦                           LineCount := LineCount + 5;   {keeps track of how many lines on a page}
23  E-------------------------end;
24
25
26  B-----------------------Begin
27  ¦                         FirstPage := true;
28  ¦                         LeaveGraphic;       {display user prompts on screen}
29  ¦                         Clrscr;
30  ¦                         Center('*** TURN ON PRINTER AND ALIGN PAPER ***',1,10,80);
31  ¦                         TextColor(green);
32  ¦                         msg('press <P> to Print, <F> to list to File *, <Q> to quit print',1,13);
33  ¦                         TextColor(WHITE);
34  ¦                         msg('* File option prints numbers to a file named "TWOPARAM.NUM"',1,17);
35  ¦                         msg('on the current drive.  Browse this file off-line using DOS "type" command',1,18);
36  ¦
37  ¦  B----------------------repeat
38  ¦  ¦                        Read(kbd,ch);             {accept user input}
39  ¦  ¦                        if (ch = 'F') or (ch = 'f') or (ch = 'P') or (ch = 'p') then
40  ¦  ¦  B---------------------begin
41  ¦  ¦  ¦                       if (ch = 'F') or (ch = 'f') then
42  ¦  ¦  ¦  B------------------------begin
43  ¦  ¦  ¦  ¦                          assign(list,'TwoParam.NUM'); {open a file for writing}
44  ¦  ¦  ¦  ¦                          rewrite(list);
45  ¦  ¦  ¦  E------------------------end
46  ¦  ¦  ¦                        else
47  ¦  ¦  ¦  B------------------------begin
48  ¦  ¦  ¦  ¦                          assign(list,'LST:');   {open the printer for printing}
49  ¦  ¦  ¦  ¦                          rewrite(list);
50  ¦  ¦  ¦  E------------------------end;
51  ¦  ¦  ¦                       NewPage;
52  ¦  ¦  ¦                       LineCount := 0;
53  ¦  ¦  ¦                       a:= amin; b := bmin;  {compute 2-parameter values for printing}
54  ¦  ¦  ¦                       for j := 1 to 5 do    {step through 5 values of selected stepping parameter}
55  ¦  ¦  ¦  B------------------------begin
56  ¦  ¦  ¦  ¦
```

```
                                for k := 1 to 50 do   { smoothly move through the other parameter}
                                begin                                    {range in 50 increments}

                                  for i:= 1 to Order + 1 do    {compute each coefficient value}
                                  begin
                                    infix_to_polish(InfixArray[i],polish);
                                    Compute_Polish(polish,a,b,EvalArray[i]);
                                  end;

                                 RootFinder(Order,EvalArray,RealPart,ImagPart,0,0); {find eq roots}
                                 if stepA then writeln(list,'  ',B:10:3)
                                        else writeln(list,'  ',A:10:3);
                                 LineCount:= Linecount + 1;
                                 For I:=1 to Order do       {write Roots to page printer}
                                 begin
                                   writeln(list,'            ',i:2,'     ',RealPart[i]:10:3,
                                                                          ',ImagPart[i]:10:3);
                                   LineCount := LineCount + 1;
                                   If (LineCount mod 55) = 0 then NewPage;
                                 end;

                                 if stepA then b := b + increm          {increment one parameter}
                                        else a := a + increm;
                                end; {for}

                               If stepA then     {increment stepping parameter}
                               begin
                                 a:= a + DeltaStep;
                                 b:= bmin;
                               end  {next step}
                               else
                               begin
                                 b:= b + DeltaStep;
                                 a:= amin;
                               end;
                            end; {for}
                          end;{if}

                until ch in ['F','f','Q','q','P','p'];
                close(list);
                EnterGraphic;    {brings back graph to screen}
                swapscreen;
              end;

                        {***************** END of PrintGraphData Procedure ************************}
```

LIST OF REFERENCES

1. Gerald, C. F. and Wheatley, P. O., Applied Numerical Analysis, 3rd ed., pp. 27-31, Addison-Wesley Publishing Company, 1984.

2. Graham, N., Introduction to Computer Science: A Structured Approach, 2nd ed., pp. 239-249, West Publishing Company, 1982.

3. Anton, H., Elementary Linear Algebra, 4th ed., p. 292, John Wiley & Sons, 1984.

BIBLIOGRAPHY

Astrom, K. J. and Wittenmark, B., Computer Controlled Systems Theory and Design, Prentice-Hall, Inc., 1984.

Borland International, Inc., Turbo Pascal Reference Manual, Version 2.0, 1983.

Borland International, Inc., Turbo Graphix Toolbox Reference Manual, Version 1.0, 1985.

Kuo, S. S. , Computer Applications of Numerical Methods, 1972.

Ogata, K., Modern Control Engineering, Prentice-Hall, Inc., 1970.

Tremblay, J. P. and Sorenson, P. G., An Introduction to Data Structures with Applications, McGraw-Hill Book Company, 1976.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Arlington, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 62 Naval Postgraduate School Monterey, California 93943	2
4.	Professor G. J. Thaler, Code 62TR Naval Postgraduate School Monterey, California 93943	15
5.	Professor A. Gerba, Code 62GZ Naval Postgraduate School Monterey, California 93943	1
6.	Professor D. E. Kirk, Code 62KI Naval Postgraduate School Monterey, California 93943	1
7.	Professor R. D. Strum, Code 62ST Naval Postgraduate School Monterey, California 93943	1
8.	Professor J. H. Duffin, Code 62DN Naval Postgraduate School Monterey, California 93943	1
9.	Professor H. A. Titus, Code 62TS Naval Postgraduate School Monterey, California 93943	1
10.	LT. Roy L. Wood Route 1, Driftwood Cove #20 Jefferson, Texas 75657	1
11.	Park, Pal Man SMC 2400 Naval Postgraduate School Monterey, California 93943	1

12. Mr. T. M. Slaughter 1
2028 Green Street
San Francisco, California 94123

18070 2
Sm

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

thesW82134
Microcomputer based linear system design
3 2768 000 75763 7
DUDLEY KNOX LIBRARY